DATA BASE

concetti e disegno

Roberto
Doretti

GRUPPO
EDITORIALE
JACKSON

DATA BASE
concetti e disegno

Roberto
Doretti

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

© Copyright per l'edizione originale
Gruppo Editoriale Jackson - Aprile 1985

COPERTINA: Silvana Corbelli
GRAFICA E IMPAGINAZIONE: Cristina De Venezia
COORDINAMENTO EDITORIALE: Daria Gianni
FOTOCOMPOSIZIONE: CorpoNove s.n.c. — Bergamo
STAMPA: Grafika '78 - Via Trieste, 20 - Pioltello (MI)

Tutti i diritti sono riservati. Stampato in Italia. Nessuna parte di questo libro può essere riprodotta, memorizzata in sistemi di archivio, o trasmessa in qualsiasi forma o mezzo, elettronico, meccanico, fotocopia, registrazione o altri senza la preventiva autorizzazione scritta dell'editore.

INDICE

Prefazione V

CAPITOLO 1

Archivio e informazione: concetti generali 1
1.1 — Il data base e l'organizzazione dei files 1
1.2 — L'archivio indexed-sequential 5
1.3 — Capacità informativa e legami logici 11
1.4 — Il software di gestione del data base e l'indipendenza 16
1.5 — Il modello gerarchico 21
1.6 — Il modello reticolare 25
1.7 — Il modello relazionale 28
1.8 — Data base centralizzato e distribuito 40

CAPITOLO 2

Il data base integrato 45
2.1 — Il principio di organizzazione integrata 45
2.2 — Il record ed il set 48
2.3 — Struttura gerarchica 53
2.4 — Struttura ad albero 62
2.5 — Strutture reticolari: il record connettore 69
2.6 — Strutture reticolari: la reticolare doppia 79
2.7 — Strutture reticolari: member-member 87
2.8 — Struttura ciclica e est opzione 93
2.9 — Estensione delle strutture base 100
2.10 — Il disegno 106

CAPITOLO 3

Descrizione del data base e sua realizzazione 111
3.1 — Il concetto di schema 111
3.2 — Schema DDL 113
3.3 — Schema DMCL 116
3.4 — Il concetto di subschema 119
3.5 — Subschema cobol o fortran e validazione 121
3.6 — Administration 127
3.7 — Il data manipulation language 132
3.8 — Utilities 135
3.9 — Le modalità operative tradizionali ed AD-HOC 140
3.10 — Dalle modalità operative and user ai linguaggi di IV generazione . 143

APPENDICE A
La scelta dei subschema 147

APPENDICE B
Data base e documentazione: il dizionario dei dati 159
B.1 — Dizionari di dati 159
B.2 — Realizzazione e funzionalità di un dizionario dei dati 166

Indice Analitico 181

PREFAZIONE

Nel mondo dell'informatica si parla da tempo di Data Base: la diffusione è ormai tale da farci pensare ad esso come ad uno "strumento" i cui vantaggi sono ineguagliabili. Forse ciò è evidente soprattutto nell'ambiente gestionale, tuttavia è errato ritenere che l'uso debba essere limitato a casi specifici: il Data Base, in realtà, è entrato a far parte di ogni tipo di applicazione, gestionale o scientifica, su grossi o piccoli sistemi, sì da divenire l'archivio per antonomasia.

Esiste, specialmente in lingua inglese, una letteratura abbastanza vasta e più o meno approfondita sull'argomento, ma in proposito va notato che essa è relativa soprattutto ai Data Base "piccoli" (ovvero a quelli che possono venir più o meno agevolmente utilizzati nei Personal Computer), oppure si limita a panoramiche abbastanza generali con scarsa attenzione ai modelli ed ai concetti teorici. Soprattutto è abbastanza raro trovare una trattazione (almeno in lingua italiana) in grado di introdurre in maniera utile al "disegno", passo finale di analisi (e dunque da specialista), ma la cui comprensione è sicuramente necessaria per la conoscenza delle possibili funzionalità applicative.

Forse quest'ultimo punto vale soprattutto per il cosiddetto Data Base Integrato che è quello che presenta maggiore flessibilità solutiva. Anche per questo motivo, ma soprattutto perchè esso è accettato quale standard Codasyl (v. nel testo), a questo tipo sarà dedicato un intero capitolo.

In effetti gli obbiettivi di questo libro sono più d'uno: introdurre ai concetti di Data Base vedendone le varie caratteristiche generali, introdurre alla conoscenza dei vari modelli, o tipi teorici di Data Base, analizzare un tipo specifico tra i più diffusi vedendo cosa significa progettarlo, disegnarlo e realizzarlo.

Mi si obietterà che non è poco e che l'impresa può risultare superiore alle forze di chiunque: raccogliere così tanto in un solo libro può non essere possibile! Sarà pur vero; tuttavia ogni cosa dipende dalle persone cui ci si vuole rivolgere, dal tipo di "pubblico" (sia detto, appunto, tra virgolette) che nello scrivere l'autore vede di fronte a sè in ogni momento. Una trattazione esauriente richiederebbe due o tre voluminosi tomi, ed un tempo di lettura non inferiore a diversi mesi, ma qualcosa che invece voglia suggerire idee e fornire informazioni di tipo concettuale richiede molto meno. D'altro canto il responsabile del settore EDP di una azienda non deve conoscere le funzionalità di dettaglio, non deve sapere se si codifica così o cosà: deve solo imparare cosa sia possibile fare e deve impararlo in breve tempo. Allo stesso modo, il programmatore che, sia pur con una certa esperienza, affronta il Data Base

per la prima volta, prima di imparare "come si fa" sulla sua specifica macchina, deve avere le idee ben chiare sul "cosa". Insomma deve imparare gli stessi concetti.

Parlando di "disegno" poco fa si è affermato che proprio questo è l'argomento fin'ora meno trattato; esso è, invece, di sicuro notevole interesse anche se esaminato solo per sommi capi e senza la pretesa di istituire metodologie (è così che procederemo). Non è possibile progettare un Data Base, o anche solamente "scegliere" di progettarne uno, senza sapere quali sono le possibilità solutive che il disegno stesso ci offre. Al contempo vedere cosa significhi "fare il disegno" ci permetterà di capire come si procede nella soluzione di un problema pratico: vedremo concretizzarsi in strutture ben organizzate la realtà applicativa, vedremo come "strutturare logicamente" lo stesso problema.

Si è detto che il libro è rivolto tanto al responsabile EDP che vuole informarsi, quanto al programmatore che entra per la prima volta nel mondo del Data Base. Aggiungerei, perchè no, lo studente di Informatica. Tutto ciò avrà fatto certo pensare ad un tipo di discorso per addetti ai lavori: sicuramente la lettura richiede qualche conoscenza; non, tuttavia, delle più approfondite. Si è cercato di mantenere il tenore delle argomentazioni nei termini della maggiore chiarezza e semplicità espositiva possibili, seguendo un filo logico ben preciso. Al lettore il giudizio sul fatto che l'obbiettivo sia stato raggiunto.

L'Autore

CAPITOLO 1

ARCHIVIO E INFORMAZIONE: CONCETTI GENERALI

1.1 Il Data Base e l'organizzazione dei files

Volendo tentare una definizione di *Data base* potremmo dire: *si tratta di un archivio unico e generalizzato per tutti i diversi tipi di applicazioni previste o prevedibili.* Naturalmente la definizione va ampiamente spiegata poichè implica, e lo vedremo tra breve, tutta una serie di concetti di una certa rilevanza. Tuttavia, tenendo presente per ora la sola idea di unicità e prima di affrontare analisi più dettagliate, è necessario conoscere almeno a grandi linee qual'è la storia della nascita dei *Data Base* e da quale situazione abbiano avuto origine.

La storia breve (e tutta scritta su periodi recenti) dell'informazione memorizzata in modo permanente è intimamente collegata all'evoluzione dei supporti. Non risaliremo alle schede: esse furono presto abbandonate come supporto duraturo. Lo furono non appena nacque il primo dei supporti magnetici (e fu abbastanza presto): il nastro. Esso era, ed è, facilmente conservabile, non presentava problemi di mantenimento ed eventuale riordino, era altamente affidabile, soprattutto permetteva di immagazzinare in pochissimo spazio una notevole mole di dati (*).

Con tale supporto, tuttavia, non si poteva lavorare che in modo sequenziale. Fu la nascita del disco a permettere un passo avanti decisivo: il nastro non poteva che essere percorso avanti o indietro in uno scorrimento sequenziale alla ricerca di una certa informazione, ora si poteva accedere ad essa anche direttamente, in funzione della posizione da essa stessa occupata. Contemporaneamente, grazie alla maggiore densità di registrazione tecnicamente ottenibile, cresceva la capacità del supporto.

Per avere un'idea più precisa di tale rapida evoluzione si osservi il grafico di figura 1.1., oggi, come si vede, si può giungere a memorizzare su disco quasi un miliardo di bytes. Inoltre archivi di dimensioni notevoli possono essere mantenuti in linea con costi relativamente bassi. Si dispone, in altre parole, di supporti capaci, economici e

---

(*) Forse proprio con il nastro nascevano le applicazioni che facevano uso di informazioni permanenti e catalogate, nasceva, insomma, ciò che propriamente può essere chiamato "*archivio*".

ad accesso assai veloce. D'altro canto il progresso tecnico non si è certo fermato e possiamo ragionevolmente attenderci che il futuro ci riservi ulteriori evoluzioni.

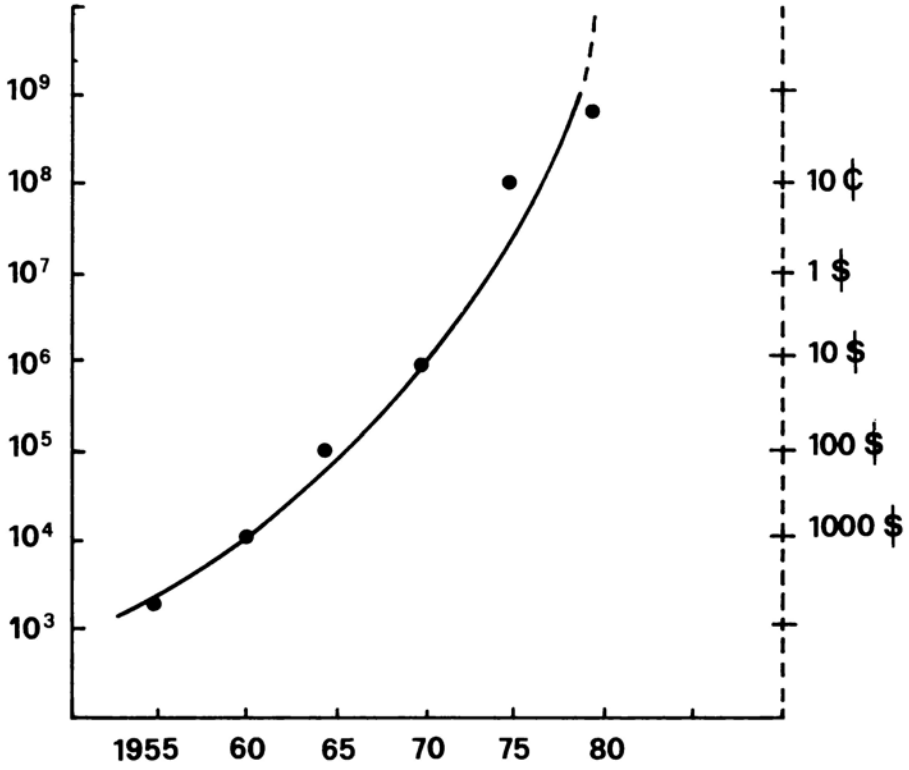


Figura 1.1.a — In ascisse si ha l'indicazione dell'anno, in ordinate (in scala logaritmica) l'ordine di grandezza del numero di bytes che possono essere contenuti in un disk-pack; i punti corrispondono ad alcuni tipo di disco tra i più diffusi nel periodo corrispondente. Sulla destra si ha un'indicazione del costo, la valutazione è in dollari ed è riferita al canone medio di affitto mensile per Megabyte.

Ora, per tornare all'evoluzione degli archivi, era ovvio che grazie a tali supporti i metodi di memorizzazione e le strutture organizzative delle informazioni divenissero sempre più perfezionati. Tuttavia, non fu immediatamente con i dischi che nacque il concetto di *Data Base*; le considerazioni che, poco alla volta, fecero sentire l'esigenza di archivi unici furono di carattere assai generale e fu l'*analisi* a condurre alla comprensione di quanto avrebbe potuto essere utile il centralizzare le informazioni.

La situazione venutasi ad istaurare vedeva la nascita di alcune organizzazioni dei files atte a soddisfare le esigenze delle applicazioni tradizionali (ma in qualche ambiente anche oggi alcune di esse rappresenterebbero una novità). Ci riferiamo ai files *sequenziali*, *relatives* ed *indexed-sequential*. Tali organizzazioni sono state definite standard, in tempi più recenti, dall'ANSI (*) ed hanno un loro importante peso in tutti i possibili tipi di applicazione. Per quanto ci riguarda vale la pena esaminarle solo quali progenitrici più o meno dirette delle organizzazioni proprie dei *Data Base*.

Non è certo il caso di soffermarci sui files sequenziali (... in generale, nè sui sequenziali su disco in particolare) se non per notare che la loro struttura non può essere la più adatta per un archivio, almeno più di quanto non lo sia quella di un nastro o di un pacco di schede.

Per quanto riguarda l'organizzazione *relative* va detto subito che essa è ammessa solo su disco. La struttura è assai semplice: ad ogni record logico il sistema fa corrispondere il numero intero ordinale della sua posizione nel file. La dichiarazione di tale numero, che è detto *chiave*, permette l'accesso diretto. Può essere stabilito un legame tra una delle informazioni contenute nel record e la sua chiave stabilendo quello che viene chiamato *algoritmo di randomizzazione*.

Si procede così: in fase di analisi si definisce l'algoritmo (una regola matematica od un qualunque altro tipo di trasformazione); nell'applicazione, una volta dichiarato il contenuto informativo sul quale basare la ricerca, si esegue l'algoritmo al fine di produrre la chiave del record cercato; a questo punto il sistema provvede automaticamente all'accesso.

Come si può immaginare, è la scelta dell'algoritmo il punto più delicato ed al proposito esistono due problemi importanti. Il primo è quello della sinonimia: due informazioni diverse possono condurre, dopo l'applicazione dell'algoritmo, a uguali valori di chiave. Ciò accade spesso se l'algoritmo è stato studiato male, ma, anche nei casi migliori, è necessario attenderci che dei sinonimi *possano* presentarsi ed è quindi necessario prevederne un'opportuna gestione. Il secondo problema è quello dello spazio: non è detto che l'algoritmo fornisca (ed in generale non lo fa) *tutti* i numeri interi. Come ovvio risultato, al momento della scrittura lo spazio riservato ai record logici, ovvero le varie *record-aree*, verrà riempito solo parzialmente (ad esempio po-

---

(*) ANSI: American National Standard Institute. Ne fanno parte tecnici ed esperti di varie case costruttrici o di vari ambienti dell'utenza. Il compito che si propone è quello di aggiornare e ridefinire periodicamente gli standard dei vari linguaggi e dei metodi di trattamento applicativo.

trebbe venir riempita la record-aerea 1, la 20, la 100, la 120 ... lasciando vuote le intermedie): se l'algoritmo è studiato male si avranno sprechi di spazio eccessivi, se è studiato bene è lecito attendersi che certi spazi restino comunque vuoti.

Visto tutto ciò, allora, quando è conveniente utilizzare la struttura del relative per un archivio?

Sicuramente quando è possibile la definizione di un algoritmo di randomizzazione che istituisca una corrispondenza uno ad uno tra informazione e posizione, minimizzando, o evitando del tutto, i problemi appunto di sinonimia e di spazio. Questo è il caso tipico dei files con caratteristiche tabellari. È il caso, ad esempio, di quei files per i quali è conosciuto il numero totale dei records, questi sono di forma e lunghezza fissa, l'applicazione vi farà riferimento come se si trattasse di una normale tabella di memoria.

Tuttavia, più in generale, si può sicuramente affermare, pur senza affrontare valutazioni sui tempi di accesso, (*) che questo tipo di archivio sarà agevolmente utilizzabile ogni qualvolta la struttura delle informazioni lo permetta ed ogni qualvolta le richieste di rapidità siano preponderanti rispetto ai problemi di spazio.

È immediato comprendere come questo tipo di files sia utilizzabile solo in casi abbastanza particolari, in applicazioni singole o per soddisfare esigenze specifiche, non, comunque, come archivio unico e generalizzato di una certa mole.

Va forse notato come tale tipo di organizzazione sia tra i più importanti da un punto di vista storico. Nata praticamente assieme ai dischi, essa fu originariamente detta *random* (**) e rappresentò la prima possibilità di archivi ad accesso sufficientemente rapido. Tuttavia i veri e propri *Data Base* si avvantaggiano di una organizzazione dei files del tutto diversa e nata con il preciso intento di rispettare da vicino le più svariate, possibili esigenze delle applicazioni, sia di permettere uno sfruttamento migliore dello spazio.

Il primo tentativo in questo senso fu fatto con gli archivi *indexed-sequential*. Anche se va detto subito che essi non nacquero in vista della istituzione di archivi unici, va però tenuto presente che essi vennero poi usati spesso proprio in quest'ottica grazie alle possibilità che offrivano. Dunque, specialmente per chi non ne abbia esperienza diretta, può essere interessante esaminare questa organizzazione più in dettaglio.

Un'altra buona ragione per far ciò può essere vista nel fatto che alcuni dei più importanti tipi di *Data Base* permettono oggi la convivenza delle organizzazioni loro proprie con quella indexed.

(*) Tempi che in un file di questo tipo sono comunque assai brevi...

(**) Esistono, ovviamente, delle differenze tra l'organizzazione *relative* e la *random* classica, ma si tratta o dell'aspetto fisico, cioè di come il file è registrato, o di alcuni aspetti applicativi. Dal punto di vista concettuale le due organizzazioni sono del tutto simili.

1.2 L'archivio Indexed-sequential

L'organizzazione *indexed-sequential* prende questo nome dal fatto che con essa è possibile trattare l'informazione tanto in modo sequenziale, quanto in modo diretto utilizzando un indice. Si parla normalmente di file *indexed* intendendo con ciò una coppia di files su disco fisicamente distinti: uno di questi prende nome *file-dati* e contiene i records logici, il secondo prende nome *file-indice* e contiene i riferimenti necessari all'accesso ai singoli record logici del file-dati. Tale riferimento è assicurato dalla presenza di una *chiave* in ogni record: questa è parte dell'informazione logica ed è univoca (nel senso che ogni record deve avere un valore diverso di chiave). Il file-indice reca l'informazione relativa alle posizioni dei vari valori di chiave nei blocchi fisici, o pagine, del file-dati.

Per chiarire tutto ciò serve forse più che altro un esempio: ammettiamo che una ditta di trasporti voglia memorizzare l'elenco completo dei suoi clienti. Ammettiamo anche che l'analisi abbia condotto alla strutturazione delle informazioni relative ad ognuno di essi secondo lo schema di figura 1.2.a. L'archivio dei clienti dovrà contenere delle informazioni rispondenti a tale struttura e l'input potrà essere del tipo descritto nella stessa figura.

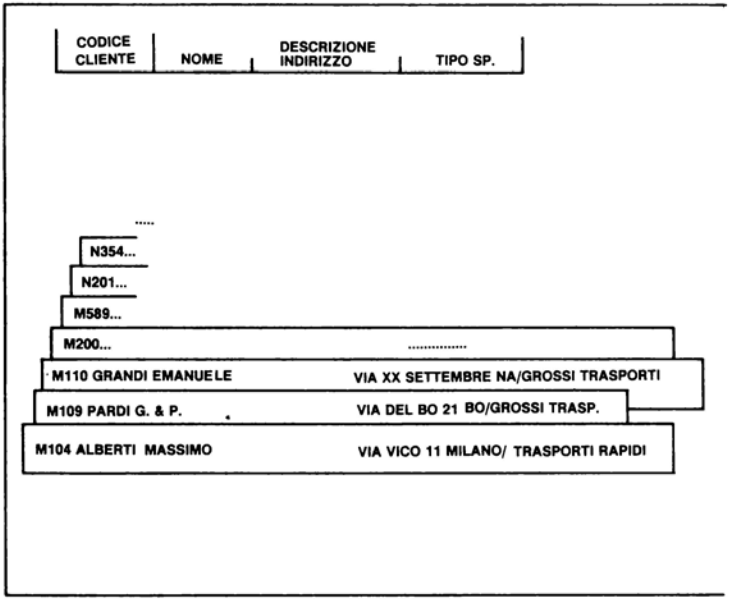


Figura 1.2.a — In alto è descritta la struttura generale del record d'archivio; più in basso un esempio del file da utilizzare come input.

Tale input è organizzato, come si vede, su di un ordinamento crescente del codice. Questo è univoco e sarà utilizzato come *chiave*. È importante notare che le informazioni in input non hanno tutte la stessa lunghezza pur rispettando la struttura generale del record, altrettanto può avvenire (ma dipende dal sistema) per i records caricati in archivio: si parla, allora, di records a lunghezza variabile.

La procedura di caricamento dell'archivio porterà le informazioni logiche nel file-dati e contemporaneamente genererà il file-indice.

Tale indice è generalmente organizzato in più livelli: nel primo livello, quello di massimo dettaglio, verrà scritto quali chiavi si possono trovare nella prima pagina del file-dati, nella seconda, terza e così via. Ai livelli superiori, di minor dettaglio, verrà scritto in quali pagine del livello inferiore si trova il riferimento a certe chiavi: insomma un indice dell'indice.

Nel caso del nostro esempio, possiamo supporre di aver eseguito un caricamento di quattro records per pagina e, si osservi la figura 1.2.b, di aver ottenuto soltanto due livelli di indicizzazione (ma in generale si arriva ad un numero maggiore).

Come è facile osservare il file può essere percorso sequenzialmente: nulla impedisce di leggere via via ogni record del file-dati secondo la sequenza logica di chiave che, nel caso della figura, e cioè dopo il primo caricamento, è anche la sequenza fisica dei records.

Come vedremo tra breve tale corrispondenza non esiste più dopo una nutrita serie di aggiornamenti dell'archivio, tuttavia la lettura sequenziale è comunque resa possibile dalla presenza dei *puntatori*: si tratta di un'informazione supplementare, automaticamente aggiunta dal sistema ad ognuno dei records al momento della loro scrittura, che consiste nell'indicazione della posizione del record di chiave immediatamente successiva. Così, nel caso il file-dati venga letto in modo sequenziale, il sistema non fa ricorso al file-indice, ma procede secondo la sequenza indicata dai puntatori. Farà ricorso, invece, all'indice nel caso di un *accesso diretto*.

Per tornare al nostro esempio, ammettiamo di voler accedere (leggere) direttamente il record di chiave *N354*, il sistema potrà procedere così: la lettura dell'indice di minor dettaglio gli permetterà di stabilire che i riferimenti a chiavi minori di *S700* si trovano in pagina 1 dell'indice di livello inferiore (o di maggior dettaglio), accedendo sequenzialmente in tale pagina stabilirà che le chiavi maggiori o uguali di *M589* e minori di *P207* si trovano in pagina 2 del file-dati. A questo punto la lettura di quella pagina del file-dati fornirà la chiave ed il record cercati.

Se in tale esempio il numero di letture (ovvero il numero di accessi alle singole pagine) può risultare paragonabile a quello che si sarebbe avuto con un trattamento sequenziale del file, va però tenuto presente che ciò accade perché la chiave cercata si trova su una delle prime pagine; se, come ovviamente accade più di frequente, il dato cercato si trova più oltre, l'accesso diretto permette di abbassare sensibilmente il numero di letture e di risparmiare un po' di tempo. Come si è già accennato tale metodo prende il nome di *accesso diretto* (o *random*) e, a volte, di *accesso con indice*.

Per quanto riguarda la possibilità di aggiornamento del nostro archivio indexed

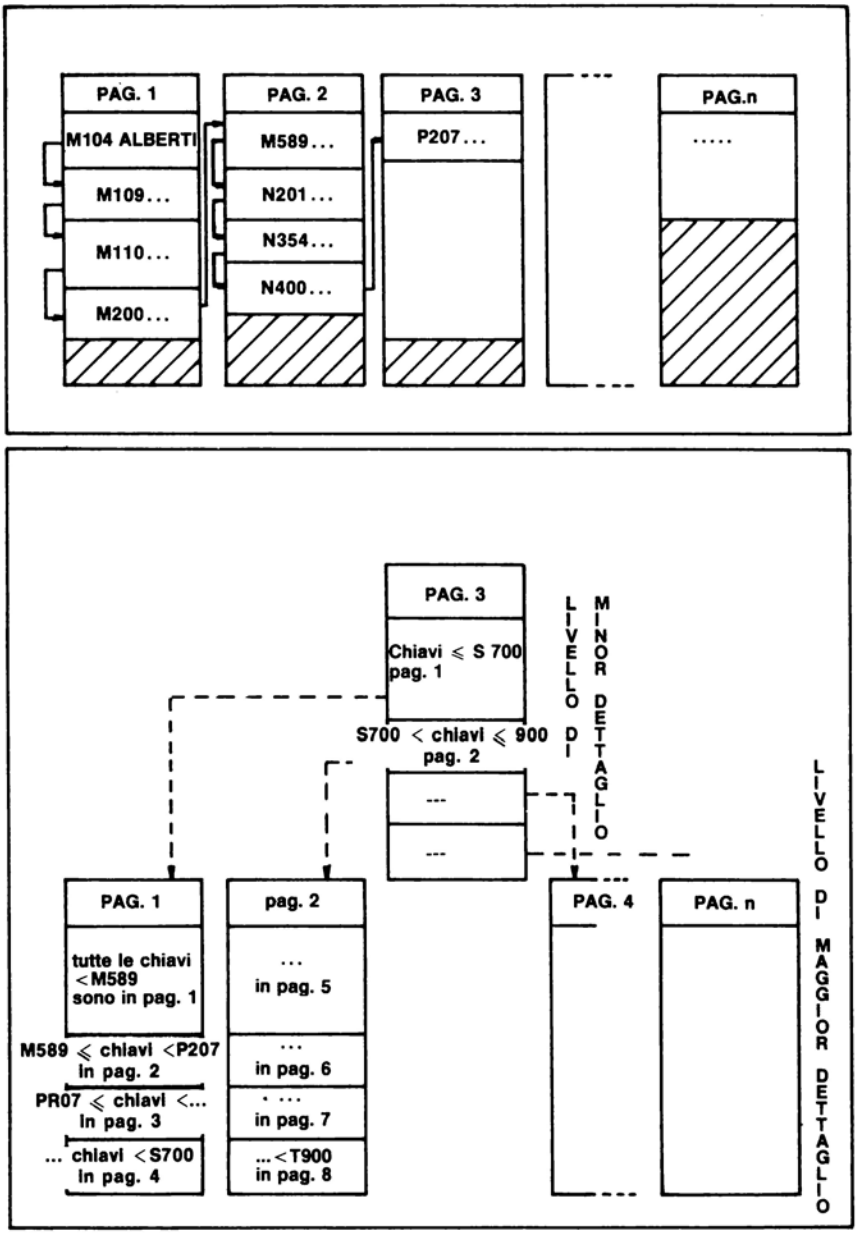


Figura 1.2.b — Schema dei files costituenti l'archivio indexed del nostro esempio dopo il primo caricamento.

(inserimento di nuovi records, modifica o cancellazione di records già esistenti) è necessario riprendere il discorso dei *puntatori*.

Come avvenga il *puntamento* di un record a quello di chiave logica immediatamente maggiore dipende strettamente dal sistema, tuttavia esistono delle caratteristiche comuni e delle funzionalità che devono comunque essere rispettate.

Innanzi tutto va notato che il primo caricamento richiede che i records siano posti in input secondo valori crescenti di chiave. Così la situazione, alla fine del caricamento stesso, sarà tale da manifestare (come nell'esempio di figura 2.b) una perfetta corrispondenza tra sequenza logica e fisica. I *puntatori* rimanderanno sempre al record immediatamente seguente. In secondo luogo va notato che la decisione di caricare un certo numero di records per pagina deve essere stata presa (almeno se siamo persone previdenti) in modo da lasciare dello spazio libero in fondo ad ogni pagina e che anche la dimensione totale del file-dati deve essere stata scelta in modo da avere delle pagine finali libere.

Fatte queste precisazioni, la funzione dei *puntatori* diviene più facilmente comprensibile: si pensi all'inserimento di alcuni nuovi records nell'archivio del nostro esempio; poniamo di voler inserire i records di chiave *M118* e *M250*. Si giunge ad una situazione in cui, come si dice, i records sono stati posti in *overflow*: essa è schematizzata in figura 1.2.c.

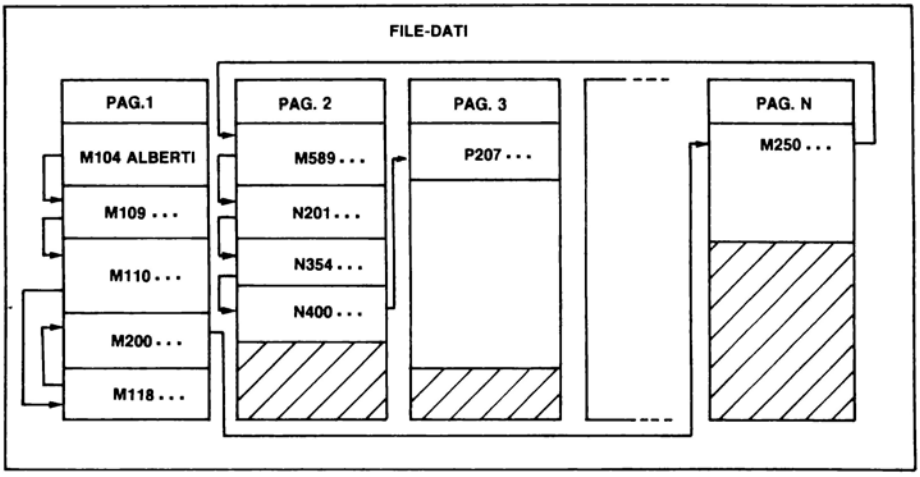


Figura 1.2.c — Il record di chiave M118 si trova in *overflow di pagina*, quello di chiave M250 in *overflow generale*.

Come è facile osservare i *puntatori* sono stati aggiornati in modo da mantenere inalterata la sequenza logica di chiave. Il record di chiave M118 non ha trovato spazio tra quelli di chiave M110 e M200, è stato però scritto nella stessa pagina ed il puntatore del record M110 è stato aggiornato in modo da rimandare alla sua posizio-

ne, a sua volta il record inserito reca un puntatore che rimanda a quello successivo (M200): si dice che il record è stato posto in *overflow di pagina*. Se lo spazio in *overflow di pagina* non risulta sufficiente, e ciò accade per il record di chiave M250, l'inserimento può avvenire in una pagina diversa, di solito una delle ultime pagine del file: si dice, allora, che esso viene posto in *overflow generale*.

La gestione degli *overflow* dipende ovviamente dal sistema (in molti casi è anche possibile lasciare libera, all'atto del caricamento, una pagina ogni n ottenendo quello che viene chiamato un *overflow locale*), ma il fatto importante è che, in ogni caso, grazie al loro utilizzo l'introduzione di nuovi records non richiede alcun intervento sul file - indice.

Esistono altri casi in cui è possibile intervenire in aggiornamento senza coinvolgere il file-indice. Forse il più importante di questi è quello delle cancellazioni. Esse possono essere di tipo *fisico*: lo spazio occupato dal record che viene cancellato è reso nuovamente disponibile (la cancellazione avviene a tutti gli effetti), oppure di tipo *logico*: nel record viene posta una segnalazione del suo *stato*, esso resta fisicamente presente, ma non può più venir letto. Nel primo caso sarà necessaria una riscrittura dei puntatori (... per le chiavi immediatamente precedente e seguente), nel secondo no (per passare da una chiave ad un'altra il sistema potrà *attraversare* un record cancellato senza tener conto della sua parte di informazione logica): in entrambi i casi i riferimenti contenuti nell'indice potranno essere lasciati inalterati.

Un ultimo fatto degno di nota a proposito degli aggiornamenti è che sia per gli inserimenti che per le cancellazioni dopo un certo periodo il file può essere divenuto inefficiente: se i records in *overflow* (generale o di pagina che sia) sono numerosi, se la sequenza logica di chiave è divenuta pesantemente differente da quella fisica, il passare da una chiave ad un'altra può richiedere in moltissimi casi, la lettura in pagine diverse, ovvero il numero di *accessi* per la ricerca di una informazione può rivelarsi eccessivo. È allora utile scaricare tutto il nostro archivio su di un file sequenziale per poi, da questo, ricaricarlo. Ciò permetterà di ottenere nuovamente la corrispondenza tra sequenza logica e fisica: essa rappresenta, ovviamente, la situazione ideale.

A questo punto non è il caso di approfondire ulteriormente l'analisi degli aggiornamenti; vi sarebbero anche altre questioni, pure importanti, ma tutte troppo intimamente legate al sistema e quindi non attinenti gli scopi che ci eravamo prefissi. Questi erano, sostanzialmente, di giungere ad una conoscenza generale corretta dell'organizzazione indexed in modo da poterne valutare l'utilità nella costruzione di un archivio. Vediamo, allora ed alla luce di quanto detto, quali vantaggi essa ci offre e quali limitazioni ci impone.

Sicuramente tra i vantaggi vanno annoverate le seguenti voci che, lo ripetiamo, si riferiscono a caratteristiche del tutto generali di questo tipo di organizzazione:

— sufficiente rapidità di accesso (almeno per files ben mantenuti, ossia scaricati e ricaricati con la frequenza che il numero di aggiornamenti richiede);

– sensibile economia di spazio su disco (qualora i dimensionamenti siano stati studiati con una certa accuratezza, le zone di *overflow* non rappresentano certo uno spreco);

– struttura generale dell'informazione assai semplice ed omogenea.

Non si tratta di vantaggi da poco, si pensi, anzi, che essi permisero la nascita dei primi veri archivi e che tutt'oggi molte applicazioni fanno ancora riferimento a *basi* di dati siffatte. Non si tratta certo, come vedremo, della situazione ottimale (del resto si sa che ogni cambiamento avviene con una certa lentezza), ma molte applicazioni, nate facendo riferimento a dei dati organizzabili in un indexed, si sono via via andate espandendo, ingrandendo a loro volta l'archivio.

Tale crescita non riguardò tanto le dimensioni, quanto la *capacità informativa* dell'archivio. Con ciò va inteso che esiste la possibilità di inserire in esso un numero abbastanza alto di informazioni *di tipo diverso*. Nell'elenco abbiamo visto come un vantaggio la semplicità e l'omogeneità dell'informazione: l'archivio indexed, si è detto, è facilmente gestibile e basta assegnare ad ogni singola informazione (ad ogni record) un *codice*, una *chiave* per poterla poi reperire, aggiornare ecc. Tuttavia, a voler ben guardare, quella voce può rappresentare anche uno svantaggio. Si pensi alla crescita dell'applicazione: poteva, e può, nascere l'esigenza di qualche informazione di tipo nuovo. Se, per tornare all'esempio precedente, oltre ai clienti della nostra ditta di trasporti, avessimo voluto memorizzare nello stesso archivio anche i dipendenti, in modo da poter fare riferimento con procedure o programmi diversi alla stessa *base* informativa, come avremmo dovuto procedere? In un caso simile la risposta è abbastanza semplice. Sarebbe bastato strutturare un nuovo record con le informazioni desiderate, assegnandogli una *chiave* allo stesso tempo congruente (come posizione e grandezza) a quella dei clienti e riconoscibile come quella di un dipendente (ad esempio la matricola del dipendente... tutte la matricole del tipo MQnn).

Come si vede avremmo dovuto soddisfare due condizioni: la nuova chiave avrebbe dovuto essere di un certo numero di caratteri ed inserita in una precisa posizione del record, inoltre essa dovrebbe essere stata scelta in modo da non risultare confondibile con le altre precedenti chiavi. Ottenuto ciò, il nostro archivio avrebbe potuto contenere due tipi diversi di informazione, ossia avrebbe avuto *capacità informativa 2*.

Come si diceva, l'evoluzione delle applicazioni ha spesso condotto ad archivi con *capacità informative* piuttosto elevate. Ciò non è, tutto sommato, un grosso male se non per il fatto che la definizione dei codici conduce inevitabilmente, prima o poi, a costruzioni artificiose, a chiavi fittizie, a terminologie confuse. Tutto ciò non può che essere fonte di errori, o quanto meno di disordine. Che dire poi se non si può (o non si vuole) stabilire una chiave congruente? Il problema diviene, ovviamente, irresolubile.

Tutto il discorso è partito dal desiderio di conglobare in un unico archivio tutte le informazioni necessarie alle nostre applicazioni. Diciamo subito che questa è, assai

spesso, una prerogativa di grande importanza. Certo nulla impedisce di stabilire un archivio di *capacità informativa 1* per ogni applicazione, ma, come vedremo tra non molto, ciò presenta gravi rischi e conduce a sprechi di tempo e di spazio. Senza approfondire, per ora, la questione si noti che il caso ottimale è quello di un archivio di *capacità informativa n* con ogni chiave indipendente dalle altre. Vedremo anche che a questa esigenza rispondono altre organizzazioni di nuovo tipo.

1.3 Capacità informativa e legami logici

Abbiamo definito la *capacità informativa* di un archivio come il numero di informazioni di tipo diverso contenute nell'archivio stesso. Vediamo di approfondire un poco l'argomento: a proposito dell'archivio indexed abbiamo esaminato un esempio di capacità informativa 2 (informazioni relative ai clienti e ai dipendenti). In generale possiamo pensare di inserire nell'archivio tutti i tipi di informazione che servono al nostro sistema informativo; la struttura definitiva delle stesse, ovviamente, dipenderà, caso per caso, dalle analogie che esse presentano e dai legami logici che tra esse intercorrono.

Esaminiamo un altro semplice esempio: ammettiamo di aver a che fare con un'azienda che ha un certo numero di clienti, dei prodotti che produce e vende, e dei dipendenti. Se ci poniamo nell'ottica di voler definire un unico archivio contenente questi tre tipi diversi di informazione, avremo a che fare con capacità informativa 3.

Ovviamente si tratta di un caso di una semplicità limite, tuttavia esso ci serve per mettere in evidenza, alcuni fatti importanti: nell'organizzazione tradizionale del sistema informativo si sarebbe dovuto procedere alla creazione di tre distinti archivi tra loro del tutto indipendenti, eventuali connessioni logiche avrebbero dovuto venir gestite da procedure ad hoc. È questo un fatto dei più importanti: le applicazioni tradizionali nascono abbastanza separate le une dalle altre e si fondano su dati memorizzati separatamente: ogni applicazione tende a far riferimento ad archivi suoi propri dotati di bassa capacità informativa.

Così, nel nostro esempio, l'organizzazione tradizionale vedrebbe le varie applicazioni rivolte separatamente ad un archivio clienti, ad un archivio prodotti, ad un archivio dipendenti. Supponiamo però che tra i diversi tipi di informazione (e quindi tra i diversi archivi) esistano delle relazioni logiche (quali, ad esempio, il fatto che certi prodotti sono rivolti a certi clienti, che certi prodotti sono venduti da quei particolari dipendenti, ecc.), è immediato comprendere come nei diversi archivi debbano essere presenti delle informazioni duplicate (fosse anche solo il codice di un articolo presso un dipendente, o viceversa) conducendo ad una situazione di *ridondanza* che presenta inevitabili problemi.

Il più evidente è lo spreco di spazio provocato dalle duplicazioni. Spesso si ritiene di disporre di volumi tali da non doverci preoccupare: nulla è più rischioso e foriero

di disastri più o meno gravi. Avremo tra breve occasione di notare che un'analisi accurata delle relazioni e degli elementi che devono entrare a far parte del nostro archivio è sicuramente necessaria e che va fatta con la dovuta attenzione alla economicità delle occupazioni.

Quello citato è un problema cui la necessità di avere delle ridondanze forzosamente ci sottopone, ma altrettanto forzosamente siamo sottoposti ad altri problemi, forse ancora più gravi, anche se meno evidenti.

Primo fra tutti il rischio di aggiornamenti incongruenti: può banalmente accadere di *dimenticare* di riportare un aggiornamento effettuato su di un archivio anche sull'altro ad esso collegato, oppure può accadere che le applicazioni siano andate evolvendosi conglobando nuove funzioni inizialmente non previste (o non prevedibili) che aggiornano certe informazioni solo su di un archivio. La cosa grave è che per mantenere gli archivi in parallelo si renderanno comunque necessarie delle elaborazioni a sé stanti, che rappresentano un qualche cosa in più, del tutto estraneo alle vere e proprie esigenze applicative di partenza. Si avranno, in definitiva, delle fasi intermedie del tutto improduttive.

L'organizzazione con Data Base, quindi con alta capacità informativa, ovvia ai problemi suddetti. Il concetto alla base di tutto è che in un archivio unico di questo tipo sono mantenuti tutti gli elementi informativi, ivi incluse le relazioni che le collegano, senza ridondanze di carattere problematico. Quest'ultimo punto significa semplicemente che nulla vieta vi siano dei dati duplicati, ma che ciò avviene senza coinvolgere chiavi d'accesso o comunque mettendoci automaticamente al sicuro dal rischio di aggiornamenti incongruenti.

Per riassumere, allora, i vantaggi offerti dall'uso del Data Base potremmo dire:

a) abbiamo a che fare con un unico flusso logico comune a tutte le applicazioni;

b) in esso non sono presenti ridondanze inutili;

c) non esiste più il rischio di aggiornamenti incongruenti;

d) le relazioni logiche sono già presenti nell'archivio e possono essere gestite automaticamente dal sistema;

e) non siamo più obbligati ad eseguire quelle fasi di elaborazione improduttiva dovute al mantenimento del parallelismo o dovute ad esigenze di riordino od apparigliamento delle informazioni.

Questo elenco riguarda caratteristiche, per così dire, intrinseche dell'organizzazione Data Base, ma esiste un altro vantaggio assai importante che va almeno accennato: si tratta dei tempi di accesso. Già è facile comprendere come la mancanza di ridondanze eviti la necessità di ripetute selezioni nella ricerca del dato voluto, ma, e questa è la cosa più importante, i metodi di gestione ed il software previsti per gli archivi Data Base permettono il maggiore risparmio, anzi è questa una delle funzionalità cui, in fase di progetto, si dà più importanza.

Torneremo a considerare questo aspetto più oltre, quando analizzeremo più da vicino i vari metodi di accesso ed i software, appunto, di Data Base.

Facciamo ancora, invece, alcune considerazioni sui legami logici: ammettiamo di aver a che fare con un archivio di capacità informativa *n* e che tali informazioni siano in relazione tra loro. Il numero massimo dei possibili legami sarà il numero di combinazioni delle informazioni a due a due cioè:

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

Definiamo il *numero di elementi m* come somma della capacità informativa e del numero di legami presenti. Nel caso siano presenti tutti i possibili legami avremo (come è facile calcolare) un numero di elementi pari a:

$$m = \frac{1}{2} n^2 + \frac{1}{2} n$$

La formula non riveste alcun particolare carattere operativo, ma ci mostra come il numero di elementi cresca con andamento parabolico in funzione della capacità informativa.

È questo un fatto tutt'altro che trascurabile e prima di tutto ci conferma in quanto già accennato a proposito delle inutili ridondanze (e cioè che il non evitarle può condurre a sprechi eccessivi), inoltre ci mostra come si possa facilmente raggiungere una notevole complessità (si pensi che con capacità informativa pari solo a 5 già si giunge ad un numero di elementi che nella nostra ipotesi sale a tre volte tanto).

Ovviamente non è detto che i legami stabiliti tra le informazioni siano sempre del tipo uno ad uno (informazione con informazione), dipenderà dalla struttura logica della relazione e da come essa viene trasposta nell'archivio. A volte basteranno anche poche, semplici, relazioni ben strutturate per raggiungere l'obiettivo di rispecchiare la situazione che vogliamo memorizzare, a volte, cioè, sarà possibile mantenere un'alta capacità informativa con un'organizzazione estremamente semplice: dipenderà dal tipo di relazione e da ciò che il software di gestione permette.

È bene dare un rapido sguardo a quelle strutture e alle loro caratteristiche generali.

Teniamo presente che l'intenzione è quella di riconoscere alcune strutture di base che permettano di ricostruire, in astratto, un'organizzazione delle informazioni poi trasponibili fisicamente sul supporto o mass-storage.

In effetti quando si pensa a come realizzare un archivio si passa attraverso varie fasi, potremmo distinguerne almeno quattro: una prima fase *informativa* in cui ci preoccuperemo di riconoscere le informazioni, di trovare le relazioni, ovvero di riconoscere i legami logici, attraverso un'appropriata serie di indagini. Una seconda nella quale ci preoccuperemo di effettuare una *riorganizzazione sintetica* di ciò che abbiamo trovato e di produrre un modello della situazione reale. Una terza fase nella

quale dovremo realizzare lo schema del modello in maniera da renderlo riconoscibile da parte della macchina. È in questa fase che assume notevole importanza riportare lo schema ad una delle strutture di base che esamineremo tra breve: ciò costituirà il primo passo nella effettiva traduzione della situazione reale. La quarta ed ultima fase riguarderà la traduzione dello schema in struttura direttamente utilizzabile nelle nostre elaborazioni.

Vediamo dunque una tipologia di queste strutture delle relazioni, lo faremo suddividendole in quattro grossi gruppi:

- gerarchiche
- ad albero
- reticolari
- tabellari

Le relazioni di tipo *gerarchico* sono forse le più immediatamente comprensibili e si presentano con notevole frequenza: lo schema è quello di figura 1.3.a. Diciamo che esiste tra due entità una relazione gerarchica quando l'una dipende dall'altra per un qualche motivo logico e/o applicativo. Diremo *dominante* l'entità a livello gerarchico superiore, *subordinata* quella dipendente. La gerarchia può espandersi su più livelli in modo tale che l'entità subordinata ad una certa dominante risultati a sua volta dominante di un'altra.

Le varie informazioni che si trovano allo stesso livello gerarchico sono accomunate dal fatto di essere tutte dello stesso tipo. Diremo che esse godono, cioè, dell'attributo comune *tipo*.

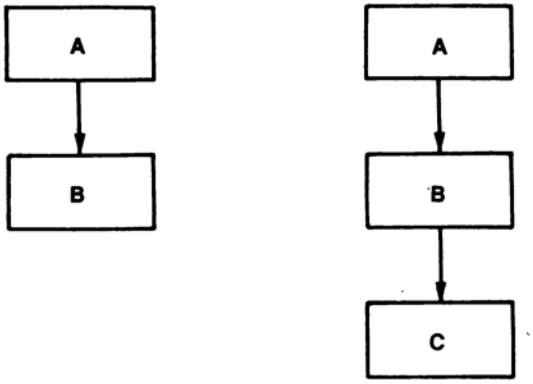


Figura 1.3.a — Relazioni di tipo gerarchico. L'informazione A è detta *dominante*, B *subordinata* nella relazione tra A e B. Nel primo caso si ha una gerarchia semplice, nel secondo a più livelli (B è a sua volta dominante su C nella relazione tra B e C).

Si dice, invece, che esiste una struttura di relazioni *ad albero* quando da una entità dominante si fanno dipendere più entità subordinate (vedi figura 1.3.b). Come si vede, si tratta ancora, in definitiva, di una gerarchia, tuttavia, questa volta, le informazioni che si trovano allo stesso livello gerarchico non devono essere necessariamente dello stesso tipo.

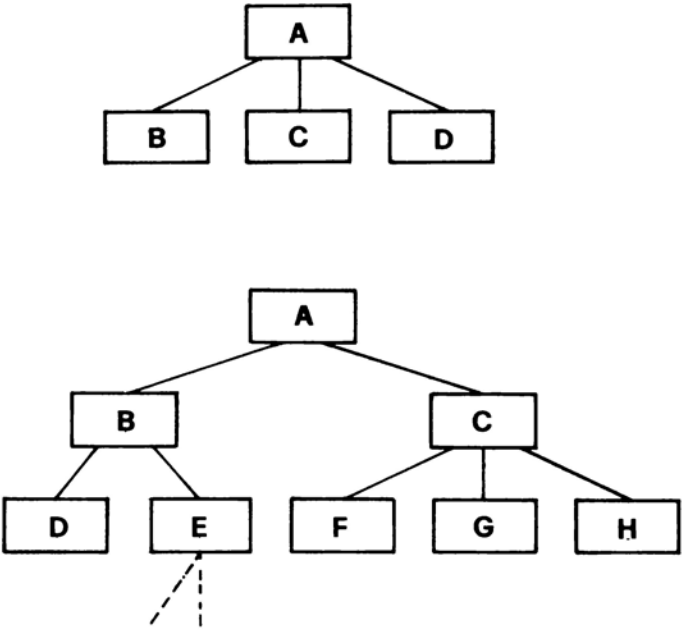


Figura 1.3.b — Relazione ad albero. Le relazioni di questo tipo sono caratterizzate dal fatto di avere, quali subordinate, entità che rappresentano tipi di informazioni diverse (due o più). Anche in questo caso si possono avere gerarchie a più livelli.

Si dice, invece, che si ha a che fare con una struttura di tipo *reticolare* quando le relazioni non rispondono ad una unica e rigida gerarchia, questa vale solo all'interno di ogni singola relazione: la struttura, vista nel suo complesso, è organizzata con un maggiore grado di libertà e risponde ad esigenze logiche assai differenziate (vedi figura 1.3.c). Può accadere che entità dominanti in una relazione lo siano anche in altre, relative a livelli gerarchici diversi; può accadere che siano contemporaneamente presenti, per la stessa entità, relazione diverse, e così via.

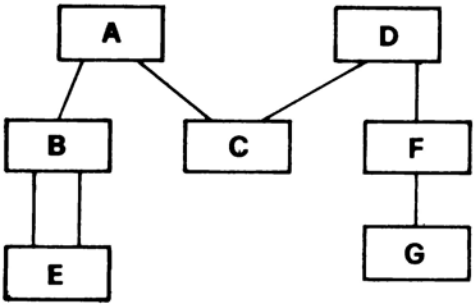


Figura 1.3.c — Un esempio di struttura reticolare: come si può osservare essa può permettere associazioni e relazioni di tipo vario. Si noti come le relazioni tra A e C e tra D e C si prestino a stabilire una relazione indiretta tra A e D.

Infine si dice che la struttura è di tipo *tabellare* quando tutti i diversi tipi di informazione sono organizzati in una tabella, o matrice, a quante dimensioni si desiderino e nella quale ogni colonna rappresenta una relazione. A proposito di questa struttura andrebbero date spiegazioni più precise e definizioni più dettagliate, essa infatti è alla base di un nuovo concetto di Data-Base, invece, lasceremo cadere il problema a questo punto per tornarvi solo più oltre quando daremo un cenno di Data Base relazionali.

Per ora ci basterà sapere dell'esistenza di quest'altro tipo di struttura e fissare invece l'attenzione sui primi tre tipi che già da soli permettono la soluzione di notevoli problemi e rappresentano l'origine delle strutture dei Data-Base oggi maggiormente diffusi.

1.4 Il software di gestione del Data Base e l'indipendenza

Prima di analizzare come le strutture delle relazioni tra le informazioni siano state realizzate in una vera e propria organizzazione logico-fisica nei vari tipi di Data Base, è bene conoscere alcuni altri concetti generali relativi al controllo delle informa-

zioni d'archivio, o, più in particolare, a quali caratteristiche funzionali debba rispondere il software di gestione.

Le varie case costruttrici, nonché varie case di consulenza, hanno prodotto (e producono) diversi tipi di software adatti al trattamento di altrettante diverse organizzazioni. Una caratteristica comune delle presenti realizzazioni, è quella relativa alla separazione tra *struttura logica* e *struttura fisica*. Tale separazione è stata ottenuta, come ora si è detto, producendo software di gestione diversi e più o meno potenti ed infatti più o meno profonda risulta essere nei vari casi. In sostanza la preoccupazione è sempre stata quella di rendere quanto più possibile indipendenti le procedure applicative dalle informazioni, dai dati fisici d'archivio.

Il software di gestione di Data Base prende generalmente nome di Data Base Management System (DBMS): esso comprende tutti i moduli eseguibili che ne permettono la generazione e l'utilizzo. È il DBMS che fa da tramite tra il sistema operativo dell'elaboratore ed il Data Base, soprattutto è ad esso affidato il compito di fare da filtro tra dato ed applicazione rendendo quanto più possibile le varie procedure indipendenti dall'aspetto fisico dell'archivio (vedi figura 1.4.a).

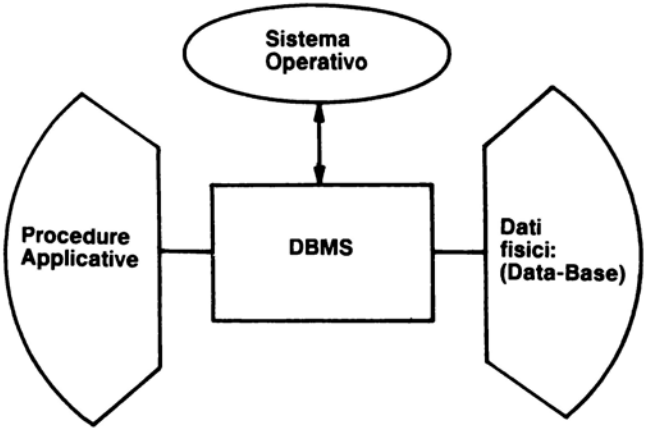


Figura 1.4.a — Il DBMS (Data Base Management System) permette di creare, amministrare, utilizzare il Data Base. Una delle sue funzioni essenziali è anche quella di realizzare un buon grado di separazione tra struttura logica e dato fisico.

Possiamo ritenere che al DBMS siano generalmente affidate delle funzioni raggruppabili in due grosse categorie:

- di realizzazione e amministrazione
- di utilizzo

del Data Base. Vale la pena approfondire un poco il discorso: nel paragrafo precedente si è parlato delle varie fasi di traduzione della situazione reale in informazione fisica. Della quarta fase, riguardante appunto la realizzazione della struttura fisica, si era dato solo un cenno. Ora va almeno precisato che il nostro elaboratore dovrà essere in grado di *conoscere* lo schema logico dei dati nonché la loro struttura fisica, dovrà, insomma, disporre di una descrizione completa del Data Base. Uno dei compiti più importanti del DBMS è appunto la traduzione di tale descrizione. Un altro sarà quello di tradurre certe descrizioni, esclusivamente logiche (qualora esse siano previste da quel particolare DBMS) che si rivolgono solamente ad una parte del Data Base, a quella più adatta a certe applicazioni e solo a quelle.

Come si vede, dunque, nelle funzioni di realizzazione sono comprese quelle che permettono di ottenere una certa indipendenza tra struttura dei dati ed applicazione. Esse avranno il dichiarato scopo di rendere quanto più agevoli possibili le modifiche dei dati (come struttura) evitando di dover sempre modificare anche le applicazioni che vi fanno riferimento.

Le funzionalità di utilizzo proprie del DBMS sono invece quelle che permettono alle applicazioni di essere attivamente eseguite. Una procedura utente sarà di solito codificata in un qualunque linguaggio evoluto, e sarà il DBMS a fare ancora da tramite tra richieste di procedura (programma in linguaggio evoluto) e intervento sul Data Base.

Per riassumere possiamo vedere le cose come nella schematizzazione della figura 1.4.b: in essa si è cercato di evidenziare come le funzioni del DBMS si rivolgono a due esigenze distinte (appunto quelle di realizzazione e quelle di carattere applicativo) e soprattutto come esse riguardino interventi cronologicamente distinti: quelle di realizzazione prima ed una volta per tutte, quelli applicativi poi, ogni qualvolta se ne presenti la necessità.

Queste caratteristiche dei software di gestione del Data Base sono, come già si è avuto occasione di notare, abbastanza generali e diffuse. Esse, in realtà, fanno riferimento al fatto che possiamo sempre vedere le informazioni da tre punti di vista diversi, o meglio, possiamo sempre pensare a tre diversi tipi di organizzazione delle informazioni:

- organizzazione logica esterna (o dell'applicazione),
- organizzazione logica,
- organizzazione fisica.

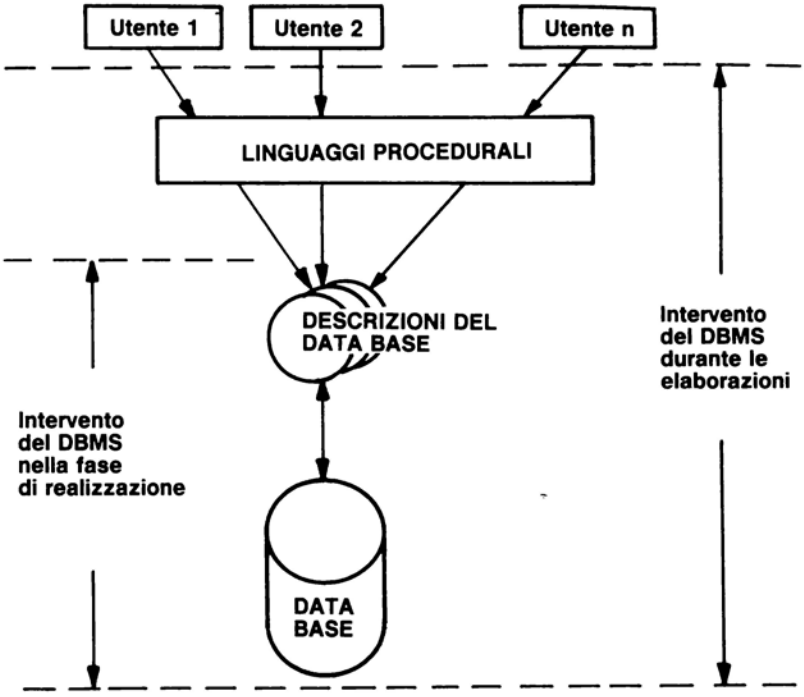


Figura 1.4.b — Schema del campo di intervento delle funzioni del DBMS.

Il primo tipo di organizzazione è quello proprio dell'applicazione: il programmatore, l'end-user, o più in generale un qualunque utente del Data Base, vede le informazioni a lui utili (per quella particolare applicazione) e solo quelle (può cioè avere una visione parziale). Si tratta di una organizzazione che può non rispecchiare affatto la realtà fisica del dato sul mass-storage pur rispettando un certo grado di coerenza con esso ed evitando incompatibilità, e che risponde proprio alle esigenze logiche dell'applicazione.

L'organizzazione logica corrisponde invece al modello concettuale del Data Base, basato su strutture organizzative come quelle del precedente paragrafo, corrisponde ad una visione totale e delle informazioni e dei loro legami, in una parola, corrisponde all'intero Data Base (e ad essa possono fare riferimento più organizzazioni esterne diverse tra loro).

L'organizzazione fisica, infine, corrisponde alla realtà effettiva del dato sul mass-storage. Si tratterà di *indici*, *chiavi*, *pointer* (vedremo di che si tratta), di occupazioni

di spazio, di quantità e via dicendo. Insomma tutto ciò che riguarda il mezzo fisico di realizzare l'organizzazione logica nonché le quantità ed i volumi coinvolti.

La distinzione tra i tre tipi di organizzazione è assai importante: gestita in modo completo dal DBMS essa permette di realizzare la succitata separazione tra dato e struttura fisica, l'indipendenza tra l'applicazione e l'archivio.

In linea di principio ogni software di gestione dei Data Base è stato realizzato in modo tale da permettere almeno due tipi di *indipendenza*, atti a salvaguardare le applicazioni da troppo frequenti modifiche o l'archivio da laboriose ristrutturazioni.

Per fare un esempio: è assai importante che se col passare del tempo il Data Base richiede delle modifiche (il caso è assai frequente) quali l'aggiunta di nuove strutture informative o di nuove relazioni, debba essere possibile intervenire, modificandolo senza per questo dover perdere, o anche solo modificare, tutte le applicazioni che nella situazione precedente erano attive e funzionanti. I due tipi di indipendenza sono:

- indipendenza logica
- indipendenza fisica.

Il primo tipo di indipendenza vuole che la organizzazione logica delle informazioni possa venir modificata lasciando intatte le applicazioni. Ciò è realizzato praticamente in tutti i DBMS esistenti e di solito è ottenuto attraverso la presenza di una sorta di *filtro* gestito dal DBMS stesso: questo è in grado di prevedere organizzazioni logiche anche tra loro assai diverse, a fronte di una stessa organizzazione esterna.

Il secondo tipo di indipendenza significa che viene resa possibile la modifica della organizzazione fisica pur lasciando intatta l'organizzazione logica (e di conseguenza possono rimanere intatte organizzazione esterna ed applicazioni): a fronte di una unica organizzazione logica possono esistere organizzazioni fisiche diverse.

È bene porre attenzione al fatto che sempre l'indipendenza è relativa alla possibilità di modifiche pur mantenendo intatte le applicazioni e che quanto detto a proposito dell'esistenza di un unico tipo di organizzazione, a fronte di altre diverse, può valere anche all'inverso nella gran parte degli esistenti DBMS (vedremo in che senso quando parleremo di *schema* e *sub-schema*).

Un altro fatto al quale è bene porre attenzione è la possibilità di cambiamenti: non è sempre possibile eseguire una qualsivoglia modifica. Ad esempio l'indipendenza logica non assicura, ovviamente, la possibilità di togliere dal Data Base delle informazioni di cui una certa applicazione si serviva, senza doverla a sua volta modificare. Se tale fatto è scontato, va detto però che esiste una più o meno vasta casistica di modifiche permesse e non permesse per ciascun DBMS e che anzi è proprio tale casistica che fornisce una misura della indipendenza raggiunta da quel DBMS e, dunque, della sua potenza.

La realizzazione pratica, come si è sopra accennato, è affidata ad una serie di *filtraggi* delle informazioni, sarebbe meglio dire ad una opportuna gestione (da parte del DBMS) di varie descrizioni corrispondenti ai tre tipi di organizzazione delle infor-

mazioni: esterna, logica e fisica. Tali descrizioni prendono il nome di *schema* e *subschema* e ne parleremo più oltre. Prima è bene tornare alle strutture informative per analizzare come la loro realizzazione abbia dato luogo a tipi diversi di Data Base, o, come spesso si preferisce dire, a *modelli* diversi.

1.5 Il modello gerarchico

Le strutture che abbiamo analizzato nel paragrafo 1.3, si è detto, conducono a strutture di Data Base sostanzialmente diverse: sulla base di tali strutture sono nati (e sono attualmente presenti sul mercato) *modelli* diversi.

L'effettiva validità e potenza di un DBMS dipende oltre che dal rispetto di certe esigenze e funzionalità generali, quali l'indipendenza tra informazione ed applicazione, anche dal modello stesso al quale esso fa riferimento.

Tratteremo brevemente di tre modelli:

- gerarchico
- reticolare
- relazionale

per analizzare le principali caratteristiche nonché i vantaggi e gli svantaggi. Del reticolare avremo modo di riparlare: trattandosi del modello attualmente più diffuso e di quello adottato dal Codasyl (vedi Capitolo II) ne esamineremo un esempio concreto.

L'ordine in cui abbiamo elencato i tre modelli corrisponde, a grandi linee, all'evoluzione dei prodotti. Già si è notato come le strutture informative più semplici ed immediate siano quella gerarchica e quella ad albero. In effetti la nascita dei primi Data Base, propriamente detti, era basata su tali strutture che noi vedremo comprese nel modello gerarchico (come si è già avuto modo di dire la struttura ad albero non è che un'estensione della gerarchia in cui le informazioni subordinate possono essere di tipo tra loro diverso).

Modello gerarchico dunque significa la realizzazione, gestibile dal DBMS, delle strutture informative gerarchica e ad albero: ogni *entità* è vista come *record logico* ed il legame logico è rappresentato memorizzando il livello gerarchico, e quindi la relazione con altri eventuali record, accanto all'entità stessa. Ogni entità dominante può essere in relazione con una o più subordinate, ogni subordinata con una ed una sola dominante.

Per meglio comprendere il senso del modello gerarchico vediamone un semplice esempio: ammettiamo di voler memorizzare quali articoli vengono prodotti da ogni singolo settore della nostra azienda e quali sono i clienti che via via li hanno ordinati. Un disegno che realizzi tale situazione può essere quello di figura 1.5.a: ogni settore aziendale è memorizzato su un record dominante, gli articoli sono rappresentati da record dipendenti rispetto ad ogni settore di produzione, ma sono a loro volta dominanti rispetto ai record cliente-ordine. Un disegno che schematizzi più da vicino la memorizzazione fisica corrispondente, è quello di figura 1.5.b.

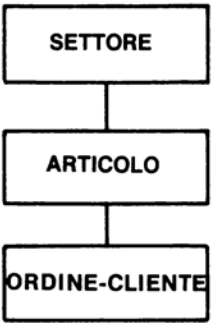


Figura 1.5.a — Modello gerarchico a più livelli: ad ogni settore possono corrispondere più articoli, ad ogni articolo più ordini. Si tratta di una schematizzazione abbreviata che evidenzia con immediatezza la natura del problema e che può essere trasposta fisicamente sul supporto di memorizzazione in modo semplice.

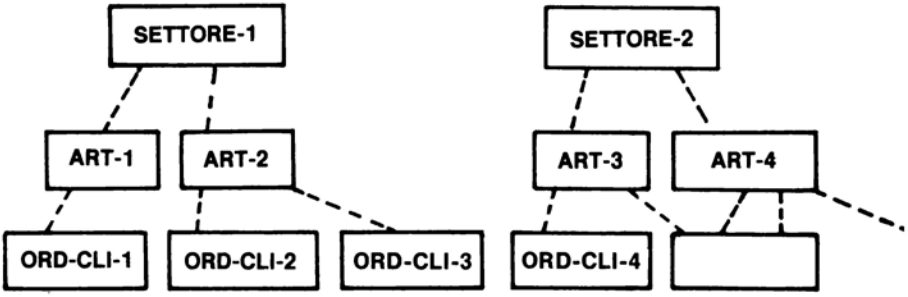


Figura 1.5.b — Lo schema di dettaglio del modello gerarchico: la realizzazione fisica di tale schema dipende dal prodotto, tuttavia in ogni caso deve poter esistere un legame uno-molti dall'alto al basso, uno a uno dal basso all'alto.

Una delle caratteristiche importanti di questo modello è la semplicità di ricerca: se volessimo selezionare certi articoli basterebbe scegliere l'opportuno settore per avere facilmente a disposizione tutti quelli che ci interessano: l'accesso ad un certo articolo è automaticamente assicurato dalla gerarchia stessa. Oppure, qualora si desideri, una volta eseguito l'accesso ad un certo articolo, conoscere a quale settore esso appartenga, è immediato risalire al suo dominante. In altre parole: i problemi che, per loro natura logica, sono di tipo gerarchico vengono trasposti, con questo modello, in maniera naturale ed immediata permettendo una gestione altrettanto immediata e semplice. Ne consegue una notevole semplicità dei comandi da usarsi nelle applicazioni, una facilità nello sviluppo del software applicativo difficilmente eguagliabile ed una visione generale delle strutture difficilmente equivocabile.

Naturalmente esiste l'altra faccia della medaglia ed accanto a questi, che abbiamo elencati come vantaggi, vi sono altri aspetti, assai più problematici. Sono questi ultimi che hanno determinato l'appesantimento e la complicazione di certi DBMS relativi a questo modello ed, in definitiva, l'evoluzione verso nuove strutture.

Un primo svantaggio va visto nella criticità di alcune operazioni: ammettiamo di dover accedere ad un certo articolo senza disporre di alcuna informazione se non la sua descrizione. Questa si troverà in uno dei campi del record accanto, ad esempio, ad un codice od altro. In molti casi il DBMS permette un accesso diretto al record, ed esempio proprio per mezzo del codice, entrando direttamente su quel tipo entità senza intervenire sugli altri livelli della gerarchia. Tuttavia, e proprio qui sta il punto, in assenza di quell'informazione (il codice) non vi sarà altra possibilità che un accesso settore per settore a uno scorrimento di ogni suo articolo fino al riconoscimento di quello cercato, nella migliore delle ipotesi sarà possibile uno scorrimento sequenziale di tutti gli articoli. Intervenire su quel solo livello della gerarchia può già rappresentare un vantaggio, ma lo scorrimento sequenziale rimane inevitabile con l'ovvio appesantimento della procedura applicativa.

Un altro svantaggio del modello riguarda invece una questione più tipicamente logica e coinvolge la necessità di duplicazione delle informazioni. Prendiamo in esame una parte del disegno del nostro esempio, quella riguardante gli articoli e gli ordini come in figura 1.5.c: se in uno stesso ordine, come è presumibile, sono elencati più articoli, sotto ognuno di questi dovrà trovarsi ripetuto quello stesso ordine. Se, d'altra parte, scegliamo di rendere dominante l'ordine, in modo da farlo comparire una volta sola, saranno gli articoli a dover essere ripetuti.

Il problema ha un duplice aspetto: ridondanza e arbitrarietà. La ripetizione di certi record può portare a ridondanze eccessive, nei casi peggiori a sprechi di spazio notevoli, comunque ad appesantimenti e sensibili complicazioni nelle fasi di ricerca.

La scelta, poi, di quale entità rendere dominante è spesso arbitraria. Ovviamente dipende dalle *richieste di procedura* cui ha condotto l'analisi: se sapessimo che nel novanta per cento delle nostre applicazioni serve scorrere, dato un certo articolo, tutti gli ordini ad esso relativi, non v'è dubbio che dovremmo scegliere l'articolo quale dominante. Nel caso opposto il viceversa. Ma se le richieste ci guidano al 50% in

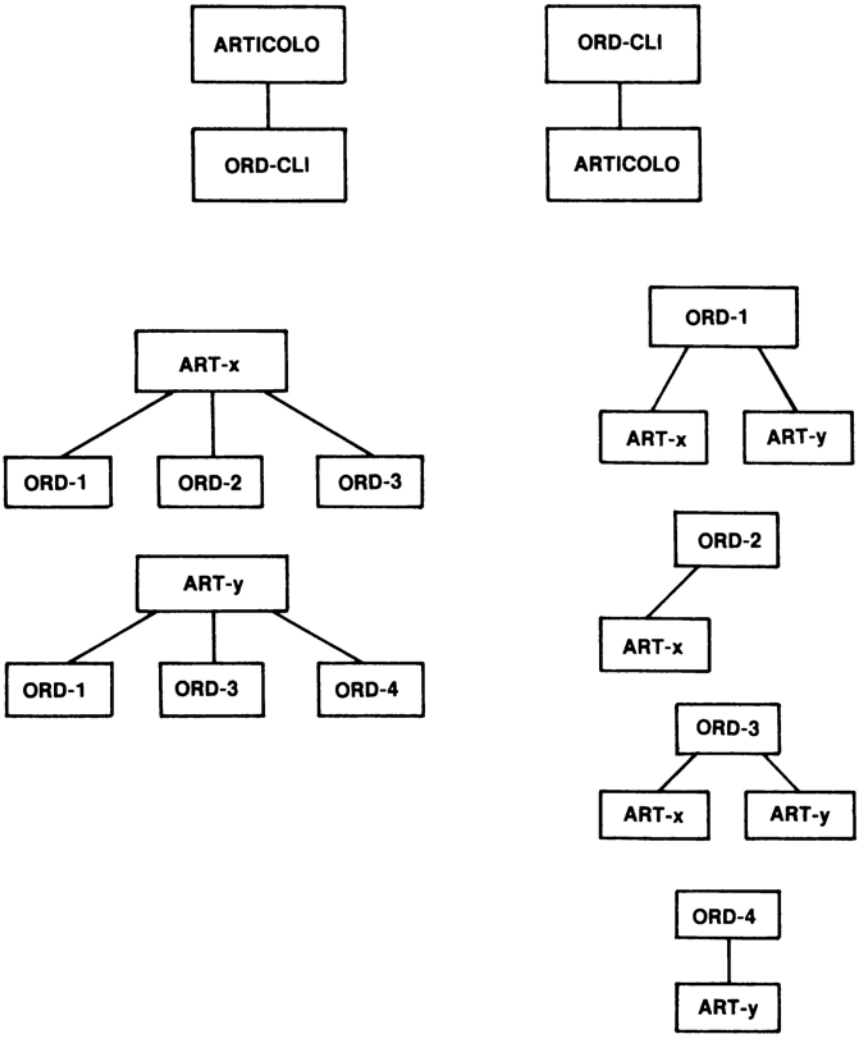


Figura 1.5.c — La scelta della relazione può rivelarsi arbitraria qualora il problema non sia, per sua natura, di tipo rigidamente gerarchico.

un senso e 50% nell'altro? Oppure, come possiamo essere sicuri che non nascano nuove richieste in futuro? La scelta non può che restare arbitraria!

Ecco dunque che nei casi in cui la logica del problema non è rigidamente gerarchica possono sorgere difficoltà notevoli faticosamente risolvibili anche con i DBMS più potenti ed efficaci.

1.6 Il modello reticolare

I problemi che il modello gerarchico presenta sono praticamente tutti superati nel modello reticolare. Esso gode di una varietà di strutture logiche assai ampia, si ricordi quanto detto nel paragrafo 1.3, e di una facoltà associativa delle informazioni assai duttile e completa. Senza dimenticare che la struttura gerarchica, essa pure, ne fa parte a pieno diritto, possiamo dire che nel reticolare una certa entità (o record) può essere vista come un *nodo* di più relazioni, ovvero possono esistere una o più entità ad essa dominanti e una o più entità da essa dipendenti.

Diamo nome *owner* al record dominante in una certa relazione, *member* al dipendente. All'insieme di tutti gli owner e di tutti i member legati da una data relazione, diamo nome *set*. In ogni set possono esistere quanti tipi diversi di member si desideri, ma un solo owner. Come nel caso del modello gerarchico la relazione è stabilita memorizzando, accanto ad ogni record logico, l'indirizzo dei record a lui collegati.

Una importante caratteristica sta nel fatto che anche i record dipendenti sono tra loro collegati (in modo da assicurare rapidi scorrimenti di tutti quelli appartenenti ad un certo owner) a formare una catena che si chiude sul loro dominante (l'owner).

I termini owner, member e set, che abbiamo adottato sono propri del Data Base Integrato che si basa su questo modello e che essendo, a nostro avviso, uno dei più potenti ed efficaci attualmente sul mercato, tratteremo in dettaglio nel secondo capitolo. Ad esso, tra l'altro, rimandiamo per un più completo esame delle caratteristiche e delle strutture. Per ora esaminiamo solo alcune delle più tipiche soluzioni possibili con questo modello notando come vengano superati alcuni dei problemi di quello gerarchico. Se riprendiamo in esame l'esempio del paragrafo precedente (ed in particolare si riveda la figura 1.5.c) possiamo verificare come con una semplice struttura (figura 1.6.a) sia possibile evitare ridondanze e appesantimenti.

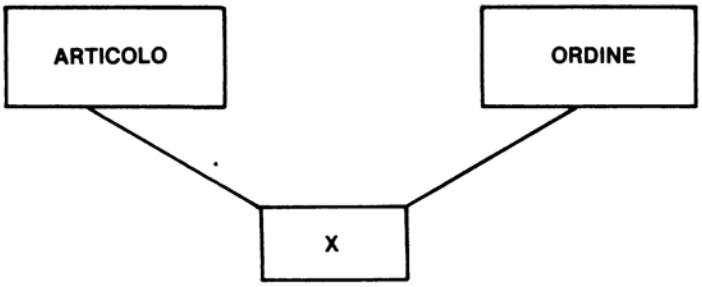


Figura 1.6.a — Il modello reticolare permette di stabilire delle connessioni tra record dominanti di tipo diverso. Il nodo rappresentato, dal record X, non comporta sprechi sensibili di spazio.

Questa volta sia l'articolo che l'ordine sono in testa alla gerarchia (sono entrambi owner) e compaiono una volta sola, il nodo, rappresentato dal record X, può contenere qualunque cosa (può essere addirittura vuoto, cioè privo di informazione logica) e costituisce una connessione tra l'ordine e l'articolo ordinato. È ovvio che in questo modo si è evitata sia la ridondanza che l'arbitrarietà della gerarchia (nel senso detto nel precedente paragrafo). Se l'evitare le ridondanze significa, e non è poco, un risparmio di spazio, soprattutto permette una più rapida ricerca e selezione e dunque efficienza dell'archivio. Per quanto riguarda poi il poter evitare l'arbitrarietà della gerarchia, va detto che ciò ci mette al sicuro dalla nascita di applicazioni troppo complesse che vorrebbero per loro natura la gerarchia inversa.

Se quella vista è forse la più tipica delle strutture del modello reticolare, va tenuto presente però che ne sono possibili varie altre. In figura 1.6.b ne sono elencate alcune: la prima è la normale gararchica, ma i record dipendenti sono di tipo diverso, la seconda realizza il cosiddetto ciclo, la terza, stabilendo due relazioni diverse tra owner e member realizza una connessione tra record dominanti, ma dello stesso tipo.

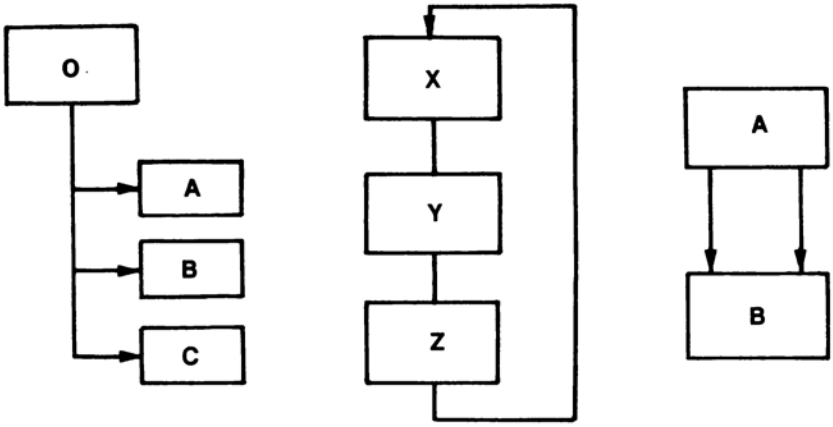


Figura 1.6.b — Alcune strutture del modello reticolare. Non tutti i DBMS del modello reticolare possono gestire tutte le strutture qui elencate, ma è loro caratteristica comune la possibilità di stabilire liberamente relazioni e connessioni.

Invece che analizzare una per una le varie strutture, ripetiamo che lo faremo nel secondo capitolo, va detto che il fatto più importante è che il disegno del nostro archivio sarà il risultato di una loro combinazione. Anzi la libertà di scelta in tali combinazioni ci dà la sicurezza di poter definire uno schema complessivo di struttura che rispecchi fedelmente le caratteristiche logiche del nostro problema.

Questo è forse uno dei maggiori vantaggi offerti dai Data Base reticolari: praticamente qualunque situazione reale può essere trasposta in termini reticolari, in quello schema di struttura, o disegno, e dunque ogni problema può essere risolto in modo pressochè ottimale (anche e soprattutto per quanto riguarda gli aspetti applicativi procedurali). Poiché non vi sono condizionamenti nel trasporre situazioni di notevole complessità, la fedeltà con cui il problema è rappresentato fa sì che anche nuove inattese esigenze possano venir risolte, per così dire, in modo automatico semplicemente facendo nascere nuove procedure lasciando intatto l'archivio.

Un altro vantaggio è sicuramente rappresentato dal fatto che un Data Base reticolare è percorribile in ogni direzione. Il fatto che si tratti di un vantaggio si spiega da sé, meno evidente è il fatto che anche ciò assicura quello che si è or ora accennato e cioè di poter far nascere procedure nuove per problemi inattesi.

Un ultimo vantaggio, questa volta più da un punto di vista generale e, se si vuole, commerciale sta nel fatto che tale modello è stato adottato come standard CODASYL e dunque i vari DBMS relativi hanno caratteristiche comuni. Esistono, anzi, degli standard sia per i linguaggi di generazione e realizzazione che per quelli di gestione del Data Base che hanno raggiunto un buon grado di evoluzione ed al contempo una diffusione notevole. Ciò non può che rappresentare motivo di sicurezza e garanzia.

Anche il modello reticolare ha, comunque, i suoi svantaggi, che, se pure non gravissimi, ci affrettiamo ad elencare: primo fra tutti la complessità ed il volume dei DBMS. Poiché esiste la possibilità di aver a che fare anche con strutture altamente complesse, il software di gestione deve essere potente e dunque, a volte di notevoli dimensioni. Ciò può presentare qualche problema su certi sistemi.

Un secondo svantaggio, sempre legato alla possibilità di strutture quanto si voglia complesse, è legato al fatto che in alcuni casi il programmare può non essere del tutto facile ed immediato come con le semplici strutture gerarchiche (ma vedremo che in alcuni casi esiste, ad esempio con il Subschema dell'Integrato, il mezzo di semplificare la visione applicativa).

Ed infine, sempre per lo stesso motivo, raramente il DBMS di tali modelli prevede linguaggi a sé stanti di interrogazione o, più in generale, di gestione immediata ed automatica. L'utente deve produrre le proprie applicazioni e il vero e proprio end-user non ha molti strumenti a disposizione. Anche in quest'ultimo caso, tuttavia, alcuni DBMS sono stati provvisti di strumenti end-user appropriati ed assai potenti. Anzi, per trarre un bilancio, relativamente a tutti i vantaggi accennati, sicuramente si può concludere che, almeno per ora, il modello reticolare è il più duttile e completo e presenta la massima sicurezza.

1.7 Il modello relazionale

Come si era accennato parlando dei legami logici, esiste una struttura delle relazioni che abbiamo denominato tabellare. Sulla base di tale struttura sono stati sviluppati concetti assai importanti che hanno preso forma nel modello *relazionale*.

Vanno subito notate due importanti caratteristiche:

a) tale modello non permette grosse economie di spazio, anzi è dichiaratamente soggetto a sensibili ridondanze;

b) esso utilizza formalismi e concetti matematici rigorosi, ma di notevole semplicità logica.

Vedremo che queste due caratteristiche comportano una serie di conseguenze da non trascurare.

Iniziamo col vedere cosa significa modello relazionale: il Data Base è costituito da una serie di tabelle diverse, ognuna assimilabile ad un file di tipo tradizionale, logicamente distinte e rappresentanti delle *relazioni*.

Prima di procedere avvertiamo che il termine relazione è qui usato in un senso diverso da quello del paragrafo 1.3: non più sinonimo di legame logico in senso generale, esso sta per tipo di corrispondenza, o ancora di legame, ma più restrittivamente è relativo al caso specifico (cioè al modello in questione). Una relazione, appunto nel modello relazionale, altro non è insomma che la tabella stessa in cui sono raccolte delle informazioni tra loro collegate.

Vediamo cosa ciò significhi per mezzo di un esempio e torniamo al caso del paragrafo 1.5. Si volevano memorizzare i settori aziendali che producono certi articoli (tralasciando per ora il problema del cliente-ordine). È ovvio che tra settore e articolo può sussistere quella che da un punto di vista logico matematico possiamo definire come relazione binaria:

SETTORE ⟷ ARTICOLO

chiaramente essa può essere espansa in una tabella come quella di figura 1.7.a. La corrispondenza non è altro che l'esigenza logica di correlare i due tipi di informazione: ogni settore con l'articolo, o gli articoli, prodotti.

Naturalmente possono esistere esigenze più complesse: non è detto che sia utile far corrispondere i soli codici, potremmo desiderare di inglobare altre informazioni quali le descrizioni, il costo degli articoli ecc. Nella tabella della figura 1.7.b si può osservare un esempio di tale caso.

RELAZIONE R1

CODICE-SETTORE	CODICE-ARTICOLO
STOF	ART10
SRBX	A200F
SRBX	A201G
STOF	A3RT

Figura 1.7.a — I codici di ogni settore sono messi in corrispondenza con gli articoli prodotti realizzando una relazione binaria. La tabella stessa è la *relazione* del modello relazionale. Si noti come certi elementi (ad esempio, nel nostro caso, il CODICE-SETTORE) possono essere ripetuti (... ad evidenziare quei settori che producono più articoli).

RELAZIONE R2

CODICE SETTORE	DESCRIZIONE SETTORE	CODICE ARTICOLO	DESCRIZIONE ARTICOLO	QUANTITÀ PRODOTTA MENSILM.	PERIODO PRODUZ. IN GIORNI	PREZZO
STOF	OFFICINA ELETTR.	ART10	MOTORE 30 WATT	1250	3	15000
SRBX	OFFICINA MECC.	A200F	TRASMISS. CINGHIA	1000	1	4500
SRBX	OFFICINA MECC.	A201G	TRASMISS. CARDAN.	700	2	10000
ABBR	LABORATORIO VERIF.	—	—	—	—	—
STOF	OFFICINA ELETTR.	ART20	MOTORE 50 WATT	1250	3	1500

Figura 1.7.b — In questa tabella la relazione è tra un numero più elevato di informazioni: essa costituisce quella che vien detta relazione multipla. Si noti che possono essere presenti voci prive di informazioni (il settore ABBR non produce alcun articolo).

La relazione R2 viene detta relazione multipla: ogni codice è correlato alla descrizione, ogni articolo al prezzo, alla quantità media prodotta in un mese, al periodo medio necessario alla produzione. Ancora possono essere presenti più volte gli stessi settori, come, d'altra parte, alcuni elementi di una riga possono mancare: è il caso del laboratorio di verifiche che non produce alcun articolo, ma che occorre sia memorizzato tra gli altri settori...

Non proseguiamo oltre: chi abbia anche solo una parziale esperienza delle tradizionali tabelle ben conosce quali siano le possibilità di gestione e utilizzo di una simile struttura informativa.

Notiamo invece come la tabella possa essere vista come una *matrice* i cui elementi sono costituiti dalle informazioni. Il passaggio, a questo punto, ai concetti logico matematici è immediato e lo verificheremo tra breve. Per ora notiamo ancora come ogni riga possa essere assai semplicemente assimilata al tradizionale record

logico. Se pensiamo, allora, alle tabelle che costituiscono il nostro Data Base, come a dei file logicamente distinti, contenenti record differenti, ecco che ne scaturisce una struttura generale assai semplice, un modello informativo assai vicino alla logica umana e dunque adatto ad una gestione altrettanto semplice.

Ma tentiamo, a questo punto, qualche definizione un po' più precisa:

- una *relazione* è un insieme di n-uple di elementi ordinati;
- si dice *grado* della relazione il numero *n* di elementi costituenti la n-upla, o riga della matrice, ognuna di esse deve contenere lo stesso numero di elementi (anche se alcuni possono essere *vuoti* o inespressi); la riga viene spesso chiamata anche *tupla*;
- si dice *cardinalità* della relazione il numero *m* di n-uple presenti: la relazione può risultare rappresentabile come una matrice $X_{m,n}$;
- ogni colonna viene detta *dominio* od *attributo.*

Nella figura 1.7.b, allora, abbiamo avuto a che fare con una relazione di grado 7 e cardinalità 5.

Naturalmente ogni n-upla deve essere diversa, almeno per un elemento, da ogni altra presente.

L'ordine con cui compaiono nella relazione non ha alcuna importanza, è invece, ovviamente, significativo l'ordine dei domini. Ogni dominio può assumere valori finiti o infiniti (i giorni di produzione dell'esempio di figura 1.7.b vanno da 1 a 31, mentre i prezzi vanno da zero ad un qualsivoglia valore).

È importante precisare che dominio e attributo non sono esattamente sinonimi come potrebbe sembrare dalla definizione. Al proposito si consideri l'esempio della figura 1.7.c in cui alla relazione R2 abbiamo aggiunto la colonna dei codici degli articoli sostitutivi (di quegli articoli cioè che possono essere offerti, a fronte di richiesta, in sostituzione di altri mancanti). Si tratta ancora degli stessi identici codici della terza colonna, ma l'uso che ne verrà fatto è sostanzialmente diverso. Si dice allora che la terza ed ottava colonna hanno lo stesso dominio, ma che esprimono attributi diversi.

RELAZIONE 3

CODICE SETTORE	DESCRIZIONE SETTORE	CODICE ARTICOLO	DESCRIZIONE ARTICOLO	QUANTITÀ PRODOTTA MENSILM.	PERIODO PRODUZ. IN GIORNI	PREZZO	CODICE ARTICOLO SOSTIT.
STOF	OFFICINA ELETTR.	ART10	MOTORE 30 WATT	1250	3	15000	ART20
SRBX	OFFICINA MECC.	A200F	TRASMISS. CINGHIA	1000	1	4500	A201G
SRBX	OFFICINA MECC.	A201G	TRASMISS. CARDAN.	700	2	10000	A200F
ABBR	LABORATORIO VERIF.	–	–	–	–	–	–
STOF	OFFICINA ELETTR.	ART20	MOTORE 50 WATT	1250	3	15000	ART10

Figura 1.7.c — Il dominio dei codici articoli è utilizzato in modo diverso (come attributo diverso) nella terza ed ottava colonna.

In altre parole, il dominio è l'insieme di tutti i possibili valori. Il vero e proprio valore in una posizione della riga è un attributo di quella n-upla.

Qualora i valori di una colonna siano tali da identificare univocamente una certa n-upla si dice che quel dominio è una *chiave*. In effetti se tutti i valori possibili sono tra loro diversi, possiamo facilmente individuare, attraverso una banale selezione la n-upla *dipendente* dal valore desiderato. Nel nostro esempio (si veda la figura 1.7.c) il codice articolo può servire allo scopo. Ovviamente possiamo pensare di definire sia il codice articolo prodotto come chiave che il codice articolo sostitutivo: dipenderà dal punto di vista sotto il quale guardiamo la relazione (in definitiva dall'applicazione): ad esempio potremmo pensare al primo come ad una chiave primaria, all'altro come ad una chiave secondaria. Qualora vi siano altri codici articolo utilizzati nella relazione come attributi diversi, si possono definire altre chiavi secondarie (quante se ne desiderino), così come si possono usare altri eventuali dominii come ulteriori chiavi secondarie.

L'esistenza delle chiavi conduce immediatamente al problema dell'utilizzo delle relazioni: tra breve vedremo che esistono alcuni tipi di operazioni, proprie del modello relazionale, che si basano su alcuni semplici principi di teoria degli insiemi. Perché tali operazioni siano possibili e permettano un certo grado di sicurezza e rapidità negli aggiornamenti del nostro Data Base, è necessario che siano rispettati certi criteri che vengono detti di *normalizzazione* delle relazioni. Per comprendere tale concetto restiamo al notro solito esempio partendo, però, da un punto di vista diverso. Ammettiamo di aver bisogno di una relazione in cui il codice settore deve essere la chiave primaria: potremmo rappresentare la relazione come in figura 1.7.d.

RELAZIONE R4

CODICE SETTORE	CODICE ARTICOLO		.....
STOF	ART10	ART20	.....
SRBX	A200F	A201G	.....
ABBR	ART10	ART20	.....

Figura 1.7.d. — La relazione non è normalizzata: per ogni codice settore esistono più articoli. Nella relazione possono essere presenti anche altre informazioni (non riportate in questa figura) come ad esempio la descrizione settore o la descrizione articolo; si noti, però che quest'ultima costituirebbe a sua volta un dominio composto.

Come si vede il dominio codice articolo, questa volta, è un dominio *composto* nel senso che esiste un vero e proprio elenco di valori costituenti l'attributo di una certa n-upla. Qui il caso è assai semplice (due soli valori), ma spesso l'attributo può essere composto da una vera e propria lista, o elenco di valori. Una situazione di questo tipo conduce inevitabilmente all'appesantimento della gestione e a vari tipi di anomalie di trattamento. Per evitare l'ostacolo la relazione deve venire normalizzata, il che generalmente consiste nello scindere la matrice originaria in altre due distinte, correlate tra loro.

In figura 1.7.e è stata realizzata una normalizzazione di questo tipo: la prima matrice è una parte della originaria, la seconda lo è a sua volta, ma contiene un attributo in più (il codice settore) che realizza la correlazione.

RELAZIONE R5

CODICE SETTORE	DESCRIZIONE SETTORE
STOF	OFFICINA ELETTR.
SRBX	OFFICINA MECC.
ABBR	LABORATORIO VERIF.

RELAZIONE R6

CODICE SETTORE	CODICE ARTICOLO	DESCRIZIONE ARTICOLO	QUANTITÀ pRODOTTA MENSILM.	PERIODO PRODUZ. IN GIORNI	PREZZO	CODICE ARTICOLO SOSTIT.
STOF	ART10	MOTORE 30 WATT	1250	3	15000	ART20
SRBX	A200F	TRASMISS. CINGHIA	1000	1	4500	A201G
SRBX	A201G	TRASMISS. CARDAN.	700	2	10000	A200F
STOF	ART20	MOTORE 50 WATT	1250	3	15000	ART10

Figura 1.7.e — Le due relazioni ridotte in prima forma normale. La normalizzazione è stata fatta a partire dal caso della figura 1.7.c.

Una condizione importante della normalizzazione eseguita in questo modo è che la parte della matrice dipendente che non riguarda la correlazione (cioè escludendo i codici settore dalla relazione) sia univoca.

Relazioni di questo tipo si dicono *in prima forma normale* ed hanno la caratteristica essenziale di non contenere attributi composti.

Osserviamo ancora attentamente la relazione R6: la vera e propria chiave è la coppia di codici settore ed articolo (per inciso, si parlerà allora di chiave composta), infatti è sulla base di tali codici che si può effettuare l'entrata in tabella ed è tale coppia che individua univocamente ogni n-upla. Tuttavia tutte le altre informazioni dipendono dal codice articolo, solo da quello e non dalla coppia di codici.

Facciamo, allora, un ulteriore passo avanti per giungere ad una ulteriore forma di normalizzazione. Vediamo prima una definizione: una relazione si dice in *seconda forma normale* quando ogni attributo dipende dalla chiave intera.

Questo non è, come si è visto, il caso della relazione R6 in cui gli attributi dipendono solo da una parte della coppia di codici. Tuttavia, eseguendo un'ulteriore scissione di quella tabella, come nella figura 1.7.f, si possono facilmente ottenere le due nuove relazioni R7 e R8, ridotte in seconda forma normale.

RELAZIONE R7

CODICE SETTORE	CODICE ARTICOLO
STOF	ART10
SRBX	A200F
SRBX	A201G
STOF	ART20

RELAZIONE R8

CODICE ARTICOLO	DESCRIZIONE ARTICOLO	QUANTITÀ PRODOTTA MENSILM.	PERIODO PRODUZ. IN GIORNI	PREZZO	CODICE ARTICOLO SOSTITUTIVO
ART10	MOTORE 30 WATT	1250	3	15000	ART20
A200F	TRASMISS. CINGHIA	1000	1	4500	A201G
A201G	TRASMISS. CARDAN.	700	2	10000	A200F
ART20	MOTORE 50 WATT	1250	3	15000	ART10

Figura 1.7.f — Riduzione in seconda forma normale: ogni attributo non chiave è funzione dell'intera chiave.

La seconda forma normale, tuttavia, ha ancora solo senso di fase intermedia: la riduzione va ancora portata avanti di un passo. Per capire come, analizziamo di nuovo la relazione R8: gli attributi descrizione, quantità prodotta, periodo di produzione, articolo sostitutivo dipendono esclusivamente dalla chiave. Facciamo però l'ipotesi (si tratta ovviamente di una semplificazione ai fini del nostro esempio) che il prezzo sia stato deciso in funzione del periodo di produzione. In tal caso quest'ultimo attributo dipende dalla chiave in modo transitivo:

CODICE–ARTICOLO→PERIODO–PRODUZ.→PREZZO

Ora una relazione si dice in *terza forma normale* quando non sia presente alcuna dipendenza transitiva, o, che è lo stesso, quando ogni attributo dipende direttamente dalla chiave e resta indipendente da ogni altro.

Ovviamente, la soluzione, per la relazione R8, consiste in un ulteriore frazionamento, come in figura 1.7.g. A questo punto finalmente ogni relazione riguarda un unico concetto base. La situazione finale può corrispondere alla figura 1.7.h in cui sono riassunte le relazioni R5, R7, R9 e R10: un paragone con la rappresentazione iniziale, la relazione R3 di figura 1.7.c, evidenzia in modo immediato l'uniformità e l'integrazione logica del modello nonché la sua semplicità e completezza.

RELAZIONE R9

CODICE ARTICOLO	DESCRIZIONE ARTICOLO	QUANTITÀ PRODOTTA MENSILM.	PERIODO PRODUZ. IN GIORNI	CODICE ARTICOLO SOSTIT.
ART10	MOTORE 30 WATT	1250	3	ART20
A200F	TRASMISS. CINGHIA	1000	1	A201G
A201G	TRASMISS. CARDAN.	700	2	A200F
ART20	MOTORE 50 WATT	1250	3	ART10

RELAZIONE R10

PERIODO PRODUZ. IN GIORNI	PREZZO
3	15000
1	4500
2	10000

Figura 1.7.g — Riduzione in terza forma normale. Si è ipotizzato che esistesse una dipendenza del prezzo dal periodo di produzione (ma non che il prezzo fosse ottenibile in maniera funzionale, come sarebbe, ad esempio, moltiplicando il numero dei giorni per un opportuno fattore).

RELAZIONE R5

CODICE SETTORE	DESCRIZIONE SETTORE
STOF	OFFICINA ELETTR.
SRBX	OFFICINA MECC.
ABBR	LABORATORIO VERIF

RELAZIONE R7

CODICE SETTORE	CODICE ARTICOLO
STOF	ART10
SRBX	A200F
SRBX	A201G
STOF	ART20

RELAZIONE R9

CODICE ARTICOLO	DESCRIZIONE ARTICOLO	QUANTITÀ PRODOTTA MENSILM.	PERIODO PRODUZ. IN GIORNI	CODICE ARTICOLO SOSTIT.
ART10	MOTORE 30 WATT	1250	3	ART20
A200F	TRASMISS. CINGHIA	1000	1	A201G
A201G	TRASMISS. CARDAN.	700	2	A200F
ART20	MOTORE 50 WATT	1250	3	ART10

RELAZIONE R10

PERIODO PRODUZ. IN GIORNI	PREZZO
3	15000
1	4500
2	10000

Figura 1.7.h — La situazione finale, dopo la riduzione in terza forma normale, può essere rappresentata da queste quattro relazioni tra loro collegate.

Forse va detto che le riduzioni comportano un certo sforzo d'analisi e che a volte si può cercare di farne a meno. Anzi, in alcuni casi, relativi ad altrettanti software di gestione, l'utente non è obbligato ad una attenta definizione delle chiavi e ad una attenta ricerca delle dipendenze. Tuttavia, le inconsistenze logiche e la presenza di ridondanze che possono essere dovute ad un difetto di normalizzazione, sicuramente producono inefficienze di trattamento (aggiornamenti lunghi e pesanti), rischi di errori ed, in definitiva, fanno perdere quei vantaggi che la semplicità e coerenza del modello invece propongono.

Nel modello descritto si rendono possibili vari tipi di operazioni. Esse sono assicurate dai cosiddetti *operatori relazionali* basati su precisi formalismi matematici. Noi tralasceremo l'approfondimento di tali aspetti matematici, ma non mancheremo di notare che gli operatori forniscono metodi assai semplici e pratici, ed al contempo potenti, di trattamento. Essi ben si prestano ad un utilizzo in linguaggi di tipo naturale, anzi hanno provocato la nascita di alcuni di essi e sono sostanzialmente destinati ad un utilizzo da parte di personale non specializzato o con scarsa esperienza DP.

Un primo operatore relazionale è quello di *selezione*: esso permette di estrarre da una relazione, o tabella, una o più n-uple che soddisfino certe condizioni dei relativi attributi generando così una seconda tabella. Ad esempio, se si desidera conoscere l'elenco di tutti gli articoli che vengono prodotti in un periodo di 3 giorni è sufficiente operare una selezione sulla relazione R9 imponendo come condizione che l'attributo desiderato sia uguale a 3: si otterrà una tabella come quella della figura 1.7.i.

Un'altra operazione interessante è quella di *proiezione*: essa permette di ottenere una nuova tabella da una data relazione, scegliendo solo certe colonne ed eliminando eventuali ridondanze. Ad esempio, volendo conoscere se esiste un rapporto tra quantità prodotta e periodo di produzione (nel nostro caso la risposta non potrà che essere no, ma la cosa non toglie alcuna validità all'esempio), potremmo eseguire una proiezione della relazione R9 sui corrispondenti attributi: si otterrebbe il risultato rappresentato ancora in figura 1.7.i.

Esiste poi la possibilità di eseguire una *congiunzione*: tale operazione può venir fatta su due relazioni che abbiano almeno un dominio comune e permette di ottenere una tabella le cui n-uple sono l'accostamento di quelle originarie (eccettuato il dominio comune che compare verticalmente una volta sola) quando gli attributi di quel dominio sono uguali. Ad esempio, una congiunzione basata sul codice settore delle relazioni R5 e R7 conduce alla tabella corrispondente di figura 1.7.i. Si noterà come questa operazione sia assimilabile alla strada inversa a quella seguita nelle normalizzazioni e come essa non sia altro che il prodotto cartesiano tra matrici.

SELEZIONE SU R9

CODICE ARTICOLO	DESCRIZIONE ARTICOLO	QUANTITÀ PRODOTTA MENSILM.	PERIODO PRODUZ. IN GIORNI	CODICE ARTICOLO SOSTITUTIVO
ART10	MOTORE 30WATT	1250	3	ART20
ART20	MOTORE 50WATT	1250	3	ART10

PROIEZIONE SU R9

QUANTITÀ PRODOTTA MENSILM.	PERIODO PRODUZ. IN GIORNI
1250	3
1000	1
700	2

CONGIUNZIONE TRA R5 E R7

CODICE SETTORE	DESCRIZIONE SETTORE	CODICE ARTICOLO	DES...
STOF	OFFICINA ELETTR.	ART10	.....
SRBX	OFFICINA MECC.	A200F	.....
SRBX	OFFICINA MECC.	A20IG	.....
STOF	OFFICINA ELETTR.	ART20	.....

Figura 1.7.i. — Le prime tre operazioni relazionali: sono basate sulle relazioni in terza forma normale dell'esempio.

Sui concetti di insiemistica sono poi basate le operazioni di unione e intersezione. L'*unione* permette di ottenere la somma priva di ripetizioni di due relazioni che abbiano gli stessi dominii. Poiché l'esempio seguito finora non si presta alla rappresentazione di tali operazioni, introduciamo due nuove relazioni (si veda la figura 1.7.j): la R11 in cui sono elencati i settori in cui è necessario che almeno una parte del personale sia altamente specializzata, la R12 dei settori in cui è preponderante il lavoro di tipo normale (per facilitare la visibilità abbiamo introdotto nuovi settori rispetto quanto fatto finora).

RELAZIONE R11

CODICE SETTORE	DESCRIZIONE SETTORE
STOF	OFFICINA ELETTR.
ABBR	LABORATORIO VERIF.
SAMM	AMMINISTRAZIONE
STPM	PROGETTAZ. E MODELLI
SINF	INFRASTRUTT. E MAGAZZ.

RELAZIONE R12

CODICE SETTORE	DESCRIZIONE SETTORE
STOF	OFFICINA ELETTR.
SRBX	OFFICINA MECC.
SINF	INFRASTRUTT. E MAGAZZ.
STRA	TRASPORTI

Figura 1.7.j — Le due relazioni descrivono: la prima i settori in cui è presente anche, o solamente, personale specializzato, la seconda i settori d'opera prevalentemente manuale.

Il risultato dell'unione di queste due relazioni può essere osservato in figura 1.7.k. In essa si può vedere anche il risultato dell'operazione di intersezione. L'*intersezione* permette di generare, a partire da due relazioni che abbiano gli stessi dominii, la tabella delle n-uple comuni ad esse.

Come è facile notare le due operazioni sono proprio l'unione e l'intersezione degli insiemi R11 e R12. È altrettanto facile osservare che esse permettono la generazione di elenchi di vario tipo atti a soddisfare qualunque tipo di esigenza (l'unione, ad esempio, ha prodotto l'elenco completo di tutti i settori).

Esiste un ultimo tipo di operazione: la *differenza*. Essa è la sola in cui l'ordine degli operandi, le relazioni, è significativo per il risultato. Permette, ancora a partire da relazioni che abbiano dominii comuni, di ottenere una tabella dalla quale sono state eliminate tutte le n-uple della prima relazione che sono presenti anche nella seconda. Così se volessimo un elenco dei settori che hanno solo personale specializzato

UNIONE DI R11 E R12

CODICE SETTORE	DESCRIZIONE SETTORE
STOF	OFFICINA ELETTR.
ABBR	LABORATORIO VERIF.
SAMM	AMMINISTRAZIONE
STPM	PROGETTAZ. E MODELLI
SINF	INFRASTRUTT. E MAGAZZ.
SRBX	OFFICINA MECC.
STRA	TRASPORTI

INTERSEZIONE DI R11 E R12

CODICE SETTORE	DESCRIZIONE SETTORE
STOF	OFFICINA ELETTR.
SINF	INFRASTRUTT. E MAGAZZ.

Figura 1.7.k — Operazioni di unione e intersezione. Tali operazioni, come dice il nome, corrispondono in tutto e per tutto a quelle della teoria degli insiemi.

DIFFERENZA R11-R12

CODICE SETTORE	DESCRIZIONE SETTORE
ABBR	LABORATORIO VERIF.
SAMM	AMMINISTRAZIONE
STPM	PROGETTAZ. E MODELLI

DIFFERENZA R12-R11

CODICE SETTORE	DESCRIZIONE SETTORE
SRBX	OFFICINA MECC.
STRA	TRASPORTI

Figura 1.7.l — Le due differenze tra le relazioni R11 e R12.

potremmo fare la differenza R11-R12 (si veda la figura 1.7.I), mentre un elenco dei settori in cui si fa solo lavoro manuale lo si sarebbe ottenuto con la differenza R12-R11 (si veda ancora la figura 1.7.I).

Per concludere questo nostro esame dei punti più interessanti del modello relazionale vanno fatte alcune rapide considerazioni di merito. Alcune sono intimamente connesse alle operazioni viste: esse hanno carattere esaustivo, nel senso che, come si è già accennato, permettono di soddisfare praticamente ogni esigenza una volta che il Data Base sia stato opportunamente strutturato in relazioni, e, per di più sono quasi completamente a carico del software di gestione. Esse, dunque, lasciano l'utente completamente libero dalla preoccupazione di gestire i legami, cosa che nelle altre strutture non è sempre possibile, e sono facilmente attivabili da comandi semplici e immediati. D'altro canto è la struttura tabellare stessa ad essere per sua natura facilmente comprensibile ed altrettanto facilmente utilizzabile.

Ancora un vantaggio del modello va visto nella alta indipendenza che, proprio per le ragioni elencate, permette: una modifica interviene solitamente su un numero limitato di relazioni, se esse sono ben normalizzate, le nuove esigenze possono essere facilmente soddisfatte.

Pochi sono gli svantaggi: una difficile analisi iniziale delle relazioni e delle dipendenze (anche di questo si era già detto) e un appesantimento delle procedure, una certa inefficienza delle prestazioni, qualora il Data Base raggiunga dimensioni notevoli. Ma proprio in quest'ultimo punto risiede la ragione della attuale scarsa diffusione del modello. Nonostante gli sforzi di molte case costruttrici, raramente fino ad oggi l'utente si è orientato ai Data Basa relazionali proprio perché essi sono, per loro stessa natura, di dimensioni notevoli. Tendono ad ingigantire facilmente con gli ovvi problemi di spazio su disco o di occupazione di memoria conseguenti.

Vorremmo concludere dicendo che se questo è sicuramente il modello che si imporrà per diffusione ed importanza in un futuro non troppo lontano (lo sviluppo tecnologico sicuramente lo permette), per ora esso non è ancora il più efficiente e consigliabile in qualunque caso.

1.8 Data Base centralizzato e distribuito

In questo esame di carattere generale sul Data Base, non poteva mancare qualche breve appunto su ciò che l'evoluzione rapidissima della componente *comunicazione* ha comportato per il gestore e soprattutto per l'utente dell'archivio.

All'inizio, come è risaputo, il Data Base era a cura del personale del *centro* e gli utenti vedevano i risultati delle applicazioni preparate dagli addetti, dalle persone che al Data Base erano anche fisicamente oltre che logicamente vicine. Una innovazione sostanziale si ebbe con l'introduzione e la successiva diffusione dei terminali (che lo sviluppo, appunto della tecnologia di comunicazione aveva permesso): na-

sceva la possibilità di fornire ad utenti più o meno remoti gli stessi servizi del centro e quindi di distribuire le funzioni in aree diverse fisicamente (anche geograficamente lontane) o logicamente (uffici o gruppi di utenza con differenti interessi e rivolti ad applicazioni distinte).

Il terminale, insomma, permetteva sia che il gestore o amministratore si allontanasse dal centro, sia, cosa forse ancora più importante, che l'utente facesse altrettanto e dunque che divenisse maggiormente autonomo e capace di scelte operative differenti, di modalità d'uso diverse, a lui più appropriate.

Ciò ovviamente ha comportato un'espansione notevole delle possibilità applicative ed un'evoluzione notevole dei sistemi di gestione (dei vari DBMS) che dovevano essere sempre più potenti e capaci di fornire funzionalità sempre più numerose e diversificate.

Il Data Base in questa ottica rimane comunque *centralizzato*, ovvero le informazioni sono pur sempre localizzate in un solo luogo, il centro, che diviene però accessibile da distanze più o meno accentuate.

Di fatto ciò non rappresenta un problema concettuale, anzi può significare un buon controllo delle stesse informazioni, una gestione più facile e corretta degli aspetti di protezione ed integrità. Tuttavia il fatto che un solo elaboratore debba garantire tutte le funzioni di gestione ed utilizzo del Data Base e contemporaneamente assicurare la comunicazione agli utenti remoti, rappresenta un limite di cui non si può non tenere conto. Questo almeno nel caso di un volume di utenza abbastanza elevato e disperso su distanze piuttosto lunghe.

La diminuzione sensibile, che la tecnologia ha prodotto, dei costi dell'hardware, ha fatto sì che si pensasse di devolvere ad un certo numero di mini-elaboratori la rete di comunicazione in modo da facilitare la gestione dei terminali. Ciò pur sempre lasciando il Data Base centralizzato in una locazione specifica. Immediatamente, però, è divenuto facile progettare il cosiddetto Data Base *distribuito*, affidando, cioè, anche il compito della gestione di una parte delle informazioni a quei mini-elaboratori. Il passo è stato inevitabile: ogni mini-elaboratore può ora lavorare su un suo Data Base e su di esso svolgere la maggior parte delle funzioni applicative, tutte le informazioni divengono accessibili da tutti i punti remoti e, dunque, il Data Base può essere visto come un tutto unico anche se fisicamente disperso in località differenti.

Esistono varie definizioni di Data Base Distribuito; forse la più nota è ancora quella data da Booth: *una base di dati distribuita esiste quando dati, posti in località diverse, sono correlati tra loro, o quando un processo in atto su di un elaboratore richiede di accedere a dati memorizzati su altri elaboratori.*

Poiché questa definizione non mette, però, bene in luce tutti gli aspetti essenziali legati alla distribuzione, citeremo una seconda definizione che è stata elaborata ed enunciata più di recente: *una base di dati distribuita esiste quando:*

a) i dati sono distribuiti fisicamente su diversi elaboratori non necessariamente dello stesso tipo;

b) l'accesso ai dati è integrato, nel senso che i dati costituiscono logicamente un'entità unica;

c) la collocazione fisica dei dati entro la rete di comunicazione è trasparente all'utente;

d) è garantita la disponibilità e la correttezza dei dati.

La definizione, maggiormente esaustiva, si spiega da sola e non sono necessari ulteriori commenti, vale invece la pena notare che partendo da essa si hanno conseguenze importanti. Prima di tutto la dimensione complessiva del Data Base può divenire incomparabilmente maggiore, in secondo luogo, fatto forse ancor più importante si ha una diminuzione sensibile del traffico di comunicazione perché la maggior parte delle applicazioni viene svolta localmente. D'altro canto il ripartire le varie funzioni su sistemi diversi offre maggiori garanzie dal punto di vista dei malfunzionamenti: la caduta di un sistema non comporta il blocco di tutte quante le informazioni di Data Base.

Tutto ciò non significa che non esistano problemi: il sistema di comunicazione può risultare complesso, i vari elaboratori devono sempre avere il compito di gestire il colloquio con gli altri ed il software necessario può risultare piuttosto pesante.

Si tenga presente che ogni utente può, a questo punto, servirsi di prodotti diversi; sia terminali che elaboratori, dotati di protocolli di colloquio anche notevolmente differenti e che i vari elaboratori possono dover lavorare anche assai pesantemente proprio per garantire la loro stessa cooperazione. L'altro problema va visto nella complessità dei meccanismi di controllo: esso risulta ripartito sui vari elaboratori senza un responsabile unico.

Nonostante questi problemi, che possono rappresentare, caso per caso, ostacoli più o meno rilevanti, negli ultimi anni le reti di elaboratori sono uscite dalla fase sperimentale e hanno raggiunto un tal grado di interesse e di diffusione che le stesse aziende di comunicazione (telefoniche e simili) hanno progettato e messo a disposizione dell'utenza diversi importanti servizi.

Non staremo ad elencarli, in questa sede notiamo solo che la tendenza attuale è quella di giungere ad una rete pubblica (la stessa rete telefonica ad esempio), unica per tutta l'utenza, il che presenterebbe gli ovvi vantaggi di una relativa economicità e di una notevole estensione.

Le prime realizzazioni di reti, erano di tipo assai più limitato: l'utente generava i propri collegamenti spesso servendosi della stessa rete telefonica pubblica che però

continuava ad essere vista non come un servizio a sé stante, devoluto alla trasmissione dati, bensì come un normale veicolo (lo stesso delle comuni conversazioni telefoniche). La crescita della *rete pubblica* interamente devoluta alla trasmissione dati per organismi diversi (ognuno con il suo Data Base Distribuito) è attualmente in atto e solo il futuro potrà dare più precise indicazioni di utilità.

Un'altra tendenza di sviluppo riguarda la *simmetria* o l'*asimmetria* delle reti, e si può sicuramente affermare che attualmente essa vede in modo più favorevole il primo tipo di rete, quello simmetrico.

Ma vediamo prima cosa significhi rete asimmetrica (vedi figura 1.8.a): in essa viene stabilita una gerarchia, un elaboratore centrale ha compiti di gestione e controllo su tutta la rete, gli altri elaboratori dipendono da esso e possono a loro volta controllare altri elaboratori. In tal modo lo scambio delle informazioni è altamente strutturato e controllato rigidamente.

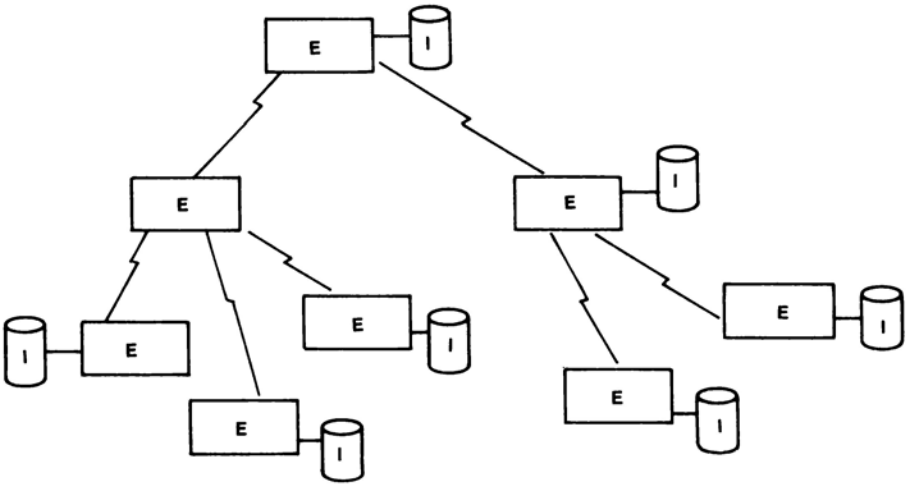


Figura 1.8.a — Rete asimmetrica. Gli elaboratori (nel disegno contrassegnati dalla lettera E) possono avere funzioni di controllo su tutti quelli che da essi dipendono. L'unione di tutti i Data Base locali (contrassegnati dalla lettera I) costituisce l'insieme informativo totale, ossia il Data Base Distribuito.

La rete simmetrica, invece (vedi figura 18.b) non rispetta alcuna gerarchia. Se da un punto di vista gestionale, la soluzione asimmetrica, proprio per il controllo che permette, offre dei vantaggi, si presentano però gravi problemi quando si tratta di far evolvere la propria rete, ad esempio con l'inserimento di nuovi elaboratori con i loro Data Base locali. In pratica, in tali casi, va ridisegnato tutto il sistema per dimensionarlo ed equilibrarlo in modo adeguato.

È proprio tutto ciò che ha condotto a puntare sulle reti asimmetriche, dove tutti gli elaborati fanno capo alla stessa rete con parità di grado e cooperano tra loro ad un uguale livello logico. Questa soluzione, ovviamente, offre maggiore autonomia agli elaboratori e sistemi locali ed una migliore adattabilità allo sviluppo tecnologico sicuramente, oggi, in fase di veloce evoluzione.

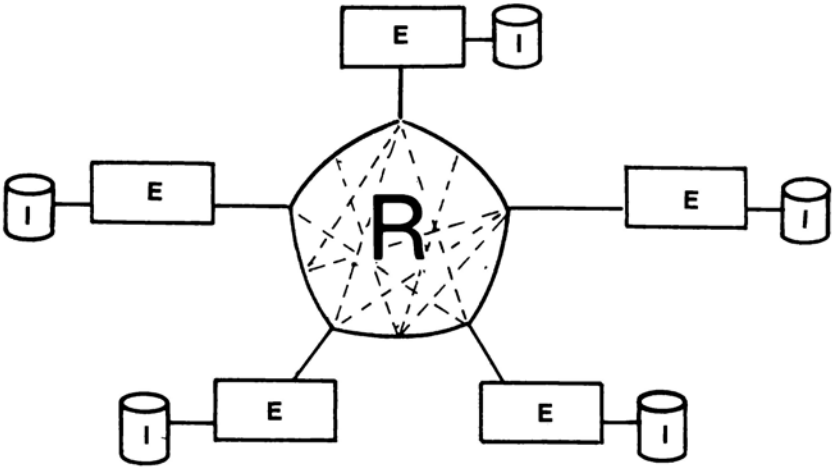


Figura 1.8.b — Rete simmetrica. Ogni elaboratore (E) è connesso alla rete pubblica (R) e coopera con tutti gli altri. Non esiste alcuna gerarchia.

CAPITOLO 2

IL DATA BASE INTEGRATO

2.1 Il principio di organizzazione integrata

Uno degli argomenti cui si è scelto di dare maggiore importanza in questo libro, è l'analisi del Data Base Integrato. Si è voluto ciò proprio perché, come si è già accennato in altra occasione, esso è un tipo di Data Base tra i più diffusi, tra i più evoluti e potenti. Il termine *potente*, in questo contesto, significa semplicemente che, a nostro avviso, se si potesse quantificare un rapporto tra i vantaggi che offre e gli svantaggi, esso si troverebbe ai primi posti di un'eventuale classifica. È ovviamente in atto un'evoluzione costante e può darsi che essa modifichi, in un futuro più o meno lontano, la scala dei valori a vantaggio di altri tipi di Data Base: il problema è proprio quanto quel futuro sia lontano.

Per motivare questa opinione che ci vede così favorevoli all'Integrato, andrebbero affrontati discorsi piuttosto lunghi, ponendo a confronto i vari tipi di organizzazione, i vantaggi di ognuno, i costi e le possibilità applicative. Tutto ciò non è nei nostri scopi. D'altro canto potrebbe essere compreso solo dopo un'analisi dettagliata sia dell'Integrato, sia degli altri tipi di Data Base. Ci basta invece notare che poiché esso si basa sul modello reticolare che, già lo si è detto, si presta alla soluzione dei più svariati problemi, dai più semplici ai più complessi, risulta essere proprio il più duttile dal punto di vista solutivo e ciò senza sprechi nei volumi di occupazione, né, di conseguenza, nei tempi di accesso.

Tra breve vedremo quali sono i principi alla base dell'*organizzazione integrata*, ma vediamone prima, in breve, la storia.

Tutto è dovuto all'invenzione di un progettista di software. Negli anni tra il 1961 e il 1962, Charles Bachman (ormai questo è un nome famoso tra gli addetti ai lavori) si occupava della progettazione di software di Data Base, con particolare attenzione al principio dell'integrazione delle informazioni, cioè della completezza sia delle relazioni che delle informazioni stesse in *archivio unico automaticamente gestibile*. L'innovazione sostanziale, o, come l'abbiamo chiamata prima, l'*invenzione*, consistette nella realizzazione di file strutturati in modo tale da rispecchiare una vasta molteplicità di esigenze logiche, istituendo legami facilmente trattabili.

Egli chiamò, appunto, *integrata* l'organizzazione di questo tipo.

Tra gli anni 1963 e 1965 venne applicata in modo più o meno efficace su vari ela-

boratori, ma il fatto importante fu che l'organizzazione integrata venne assunta come standard dal Codasyl, ovvero da quel gruppo di progettisti e studiosi che si occupava, come tuttora si occupa, di stabilire degli standard, appunto, di disegno e realizzazione del software applicativo e dei linguaggi di programmazione.

Fu nel 1967 che venne formato il DBTG (Data Base Task Group): esso ebbe il compito di stabilire, forse meglio di scegliere, un assetto formale delle possibilità applicative, delle strutture organizzative e del software di gestione dei Data Base, che potesse risultare comune agli utenti più diversi e alle case produttrici più antagoniste. La sola IBM si discostò (forse con meno piuttosto che con più merito) dalle decisioni del gruppo che attorno al 1970 stabilì che fosse l'organizzazione integrata la strada comune da seguire. La decisione, formalizzata dal Codasyl, di cui il DBTG faceva parte, venne accettata dalla maggioranza dei produttori e ciò condusse alla situazione che attualmente viviamo.

Da allora sono passati alcuni anni e, tuttavia quelle decisioni sono rimaste assolutamente valide: praticamente tutte le case produttrici hanno adottato lo standard Codasyl per il loro Data Base permettendo all'utente, da un lato, una visione della problematica assai unificata, dall'altro una grossa facilità di passaggio da un sistema ad un altro.

Tralasciando i problemi e le scelte dei produttori, nonché i fatti commerciali che costringono l'utente ad una cernita assai attenta, va tenuto presente, ancora una volta, che quel che conta è la possibilità solutiva.

Forse tornando ad una rilettura dei paragrafi relativi ai modelli, l'obiezione che il relazionale è il più facilmente gestibile, risulterebbe giustificata, ma ancora va ripetuto che per ora esso è realizzabile solo pagando un prezzo piuttosto elevato in termini di occupazione di spazio e studio e disegno iniziali.

A questo punto, comunque, val la pena passare ad analizzare cosa significhi organizzazione integrata. Essa si basa, fondamentalmente, sul principio uno a molti: se da un'informazione, per questioni di tipo logico o applicativo, dipendono altri tipi di informazione, queste ultime saranno collegate alla prima in modo diretto (realizzando, cioè, un tipo di relazione gerarchica) e, per di più, saranno tra loro collegate in una sorta di catena (che le fa comunque tutte soggette alla prima). Per parlare in termini più corretti dovremmo dire: se tra l'informazione di tipo A e quella di tipo B esiste una relazione del tipo:

$$A_i \longleftrightarrow B_{i,j}$$

dove i e j possono avere un valore finito qualunque, ogni A_i permette di leggere tutti i $B_{i,j}$, ogni $B_{i,j}$ permette di riconoscere il dominante A_i.

Da un punto di vista teorico ciò non è poco, tuttavia, quelloche forse più conta è, ancora una volta, la realizzazione pratica del principio: se A è un tipo di informazione da cui ne dipendono varie altre di tipo B, l'organizzazione integrata vede A come un record, al quale è collegato il primo, il secondo, e così via, dei B (come in figura 2.1.a); al contempo ogni B permette di risalire a quel particolare A che è origine della concatenazione.

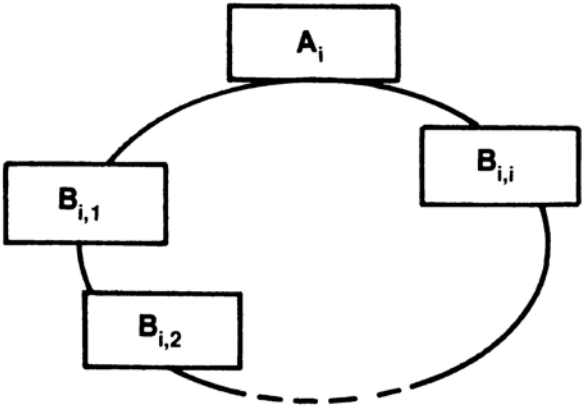


Figura 2.1.a — Da ogni record dominante A dipende un certo numero di record B, realizzando una sorta di struttura gerarchica. Ciò che è importante per il principio di integrazione è che questa struttura di base può combinarsi in vario modo con altre generando un *reticolo* informativo che realizza il modello, appunto, reticolare.

Naturalmente tutto ciò non basta alla qualifica di organizzazione integrata: va aggiunto che ogni dipendente (B) può a sua volta essere dominante in un'altra relazione con informazioni di altro tipo e che queste ultime possono a loro volta esserlo in altre relazioni realizzando, giustappunto, quel modello che (vedi paragrafo 1.6) abbiamo chiamato reticolare.

Comunque vanno esaminati più da vicino i principi organizzativi e le singole strutture che compongono il modello e, prima di tutto, vanno analizzati i principi fisici che realizzano quei legami. Lo faremo nei prossimi paragrafi proprio partendo dalle strutture più semplici e di base.

Si tenga presente che utilizzeremo la terminologia propria di un particolare prodotto (il DM IV della Honeywell), ma ciò non toglierà alcuna validità generale al discorso, nel senso che tutto quanto verrà detto potrà essere ritenuto valido, appunto con l'esclusione di alcune particolarità di terminologia, per qualunque tipo di Data Base Codasyl.

2.2 Il record ed il set

La realizzazione della semplice struttura esaminata nel paragrafo precedente (vedi figura 2.1.a) è affidata ad una serie di indirizzamenti tra i record: ogni record porta con sé l'indirizzo degli altri cui è collegato. Tale informazione accessoria (aggiunta, appunto, all'informazione logica presente nel record e gestita dal sistema) prende nome **pointer**. Non esaminiamo, per ora, come avviene in dettaglio tutto ciò, ci è sufficiente tenere presente il principio che sta alla base di questo collegamento, o relazione, tra record: esso ci servirà per affrontare alcune definizioni e precisazioni di terminologia che si rendono necessarie prima di procedere.

Un disegno schematico della situazione di figura 2.1.a viene solitamente eseguito come in figura 2.2.a: esso prende nome **shorthand** e indica che ad ongi record A possono corrispondere uno o più record B, o nessuno. Come è facile accorgersi, questo può essere un modo semplice di realizzare un tipo un po' particolare di struttura gerarchica (ma sulle strutture avremo occasione di tornare più oltre).

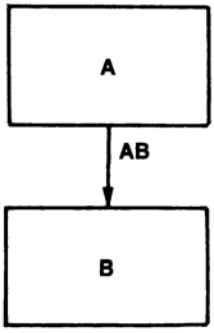


Figura 2.2.a — Shorthand: disegno schematico della relazione tra gli A e i B, o, se si preferisce, del *set* AB.

L'insieme di tutti i record A e di tutti i record B, prende nome **set** o **catena**. Ogni record A prende nome **owner** e ogni B **member**.

Ad uno shorthand come quello della figura in questione possono corrispondere varie situazioni: ammettiamo, ad esempio, di avere 5 record di tipo A e che ad ognuno di essi vengano collegati i suoi member. Si può disegnare uno schema di dettaglio come quello di figura 2.2.b: esso prende nome **longhand**.

Per motivi di comodo decidiamo di chiamare **catena singola** ogni gruppo composto dall'owner con i suoi member (ad es. A_1, $B_{1,1}$, $B_{1,2}$, $B_{1,3}$), come si noterà possono esistere catene singole vuote (si veda l'owner A_4): in tali casi il record punta a sé stesso e si parla di *catena esistente* proprio in quanto l'indirizzamento è comunque già predisposto. È evidente che ogni catena singola deve contenere un unico owner. Si tenga inoltre presente che in un set (è questa una regola fondamentale di questo tipo di organizzazione) l'owner in un set può essere costituito da un unico tipo record.

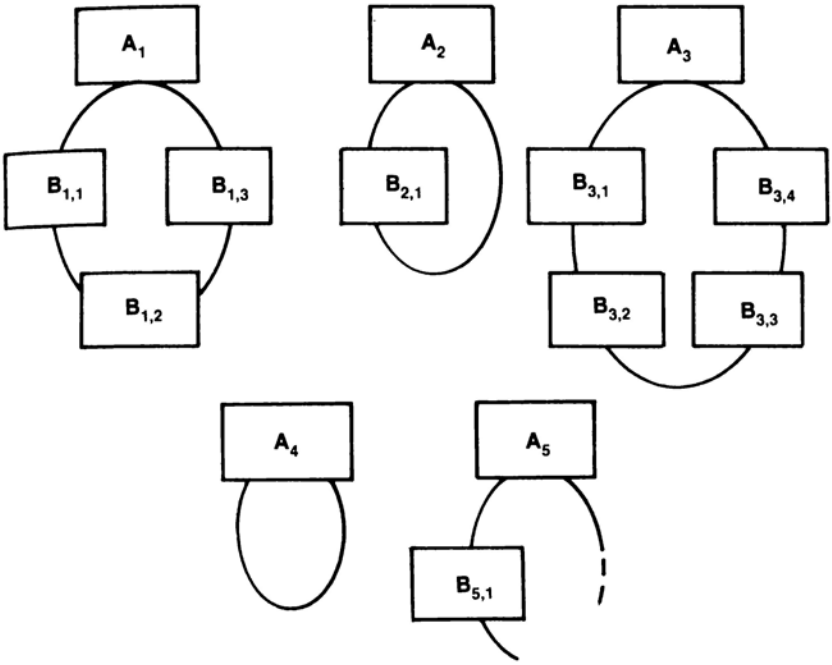


Figura 2.2.b — Longhand: il set AB visto in dettaglio. Sono evidenziate la dipendenza dei B dai relativi owner A e le relazioni tra loro, non si dimentichi, tuttavia, che da ogni B si può immediatamente risalire al suo owner.

Nel longhand i collegamenti sono disegnati in modo abbastanza efficace per quanto riguarda l'aspetto intuitivo, ma non rispecchiano del tutto l'effettiva realtà fisica. Di fatto va tenuto presente che ogni record owner dispone di due pointer (per ognuno dei set in cui partecipa) che permettono lo scorrimento della catena singola in avanti o all'indietro, o se si preferisce, permettono l'indirizzamento rispettivamente al primo o all'ultimo dei member: essi prendono nome di *pointer next* e *pointer prior*. Ogni record member dispone di tre pointer (per ognuno dei set in cui partecipa): di questi, due hanno ancora nome pointer next e pointer prior e permettono l'indirizzamento al prossimo o al precedente member, il terzo prende nome *owner pointer* e permette l'indirizzamento all'owner (direttamente, da qualunque posizione della catena singola).

Si osservi ora la figura 2.2.c: da essa risulta sufficientemente chiaro come siano realizzati effettivamente gli indirizzamenti nel caso della catena singola di owner A della figura 2.2.b. Si noterà anche che per taluni record due pointer diversi portano la stessa indicazione (è il caso del pointer prior e dell'owner pointer di $B_{1,1}$ o dell'owner pointer e del pointer next di $B_{1,3}$), ma non si tratta di informazioni sovrabbondanti (ad esempio possono permettere il riconoscimento da parte del sistema del primo e dell'ultimo record).

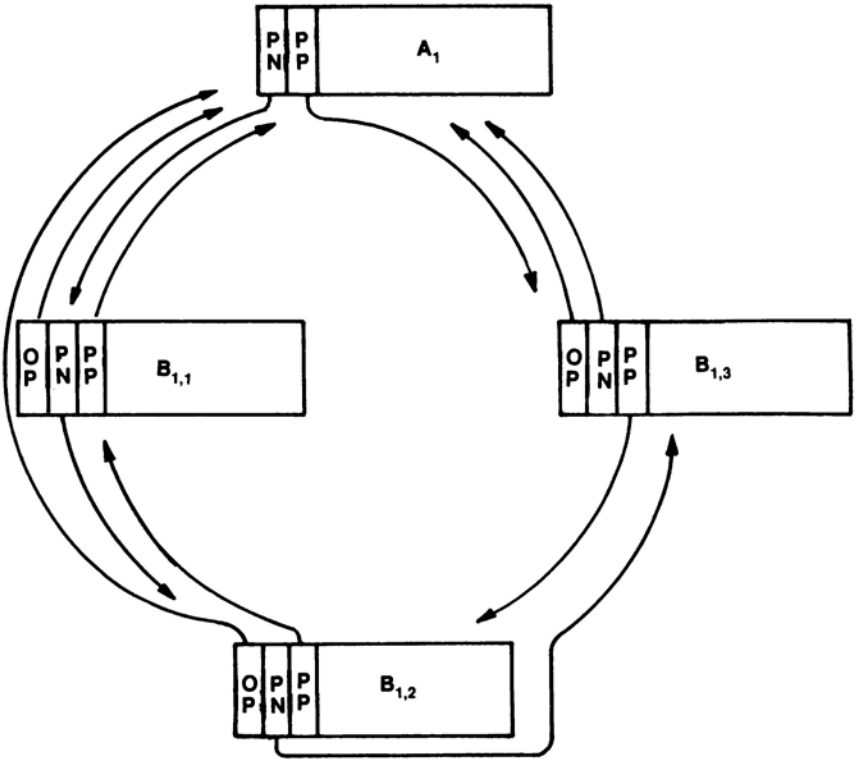


Figura 2.2.c — Pointer: l'owner dispone di due pointer, ogni member di tre. OP sta per *owner pointer*, PN per *pointer next*, PP per *pointer prior*.

Se avessimo voluto disegnare la catena singola dell'owner A_4, avremmo dovuto far sì che i pointer next e prior indirizzassero allo stesso A_4, ad evidenziare la situazione di catena singola vuota. Un altro fatto da osservare, ancora in questa figura, è la posizione dei pointer.

Essi sono stati disegnati in testa al record (come accade per molti sistemi), separati dalla informazione logica: non si dimentichi che essi costituiscono un'informazione accessoria, del tutto trasparente all'utente e gestiti automaticamente dal sistema. In effetti l'importanza di questa organizzazione sta proprio nel fatto che in un'applicazione, l'utente dovrà preoccuparsi solamente di procedere in avanti o all'indietro per fare degli scorrimenti sequenziali, oppure potrà risalire direttamente all'owner, il tutto usando un opportuno linguaggio applicativo abbastanza semplice e vedendo dunque il problema solo dal punto di vista logico.

L'indirizzamento all'altro record, la lettura del pointer ed il successivo accesso al prossimo, precedente record, ecc., rimangono del tutto a carico del sistema. Anche la creazione (strutturazione) dei pointer resta a carico del sistema e riguarda le fasi

iniziali di realizzazione del Data Base. L'utente avrà esclusivamente il compito di descrivere la struttura generale, di trasportare il disegno (in forma shorthand) in linguaggio descrittivo definendo quali record sono owner, quali member e quali sono i set cui appartengono. Avrà, in altre parole, un compito di natura, ancora una volta, logica.

Torniamo indietro di un passo: si è detto, tra parentesi, che ogni owner ha due pointer per ogni set in cui partecipa. Tre per i member. È importante notare che un record può partecipare in più set e che il conteggio deve tenere conto della partecipazione complessiva. La questione può risultare più chiara osservando lo shorthand di figura 2.2.d: si era già accennato che il Data Base Integrato realizza il modello reticolare e che ciò viene ottenuto combinando vari tipi di struttura. Ora, se vogliamo, possiamo dire che è possibile eseguire quelle combinazioni istituendo set diversi ove ci sembri più utile farlo.

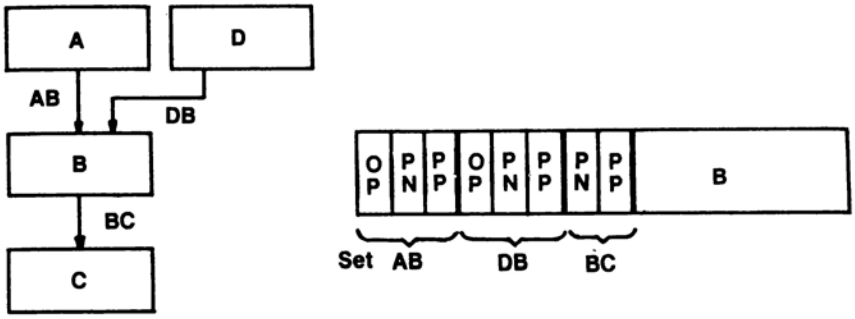


Figura 2.2.d — Un esempio della appartenenza di un record a più set: ogni record B dispone di un totale di otto pointer.

Nella figura che stiamo esaminando sono stati disegnati vari set (e battezzati con i nomi AB, BC, DB) a collegare quattro tipi diversi di record. Ancora lasciando da parte il problema strutturale ed il discorso conseguente, si noterà come il record B possa essere visto come member (nei due diversi set AB e BD), o come owner (nel set BC). Ovviamente per ognuno dei set avrà l'appropriato numero di pointer come risulta dal dettaglio a fianco dello shorthand ed in totale dovrà disporre di otto pointer.

A questo punto ci si potrebbe domandare: ma, in definitiva, ora che abbiamo visto dove stanno e a cosa servono i pointer, cosa vi viene effettivamente scritto e come avviene il vero e proprio indirizzamento?

La risposta è assai semplice. Si tenga prima di tutto presente che il file fisico su cui si trovano i nostri record sarà, ovviamente, suddiviso in blocchi fisici. Solitamente, parlando di Data Base, tali blocchi prendono il nome di *pagine*. Ora, ciò che sta scritto (anche se ciò ovviamente dipende dal sistema) nel pointer non è altro che un

valore numerico che il sistema è in grado di trasformare, attraverso un banale algoritmo, in numero di pagina e posizione (in quella pagina) del record. Tale valore numerico può venir chiamato *chiave diretta* del record o DBK (Data Base Key): esso individua univocamente il record e ne permette l'accesso da parte del sistema.

Quanto detto ben serve a chiarire un ultimo importante concetto: se la posizione fisica del record è univocamente individuabile nel modo visto, anch'essa può essere gestita autonomamente dal sistema. Ciò significa che, ad esempio, durante il caricamento la ricerca delle posizioni in cui mettere il record potrà essere ancora a carico del sistema, che seguirà opportuni criteri, ma che lascerà l'utente libero da qualunque preoccupazione che non sia, ancora una volta, di natura logica. Ovviamente ciò potrà comportare una disposizione dei reocrd nel Data Base per nulla corrispondente all'ordine logico quale apparirebbe da un longhand (si veda la figura 2.2 e). Ciò che è importante, tuttavia, è che quell'ordine fisico risulterà praticamente trasparente all'utente.

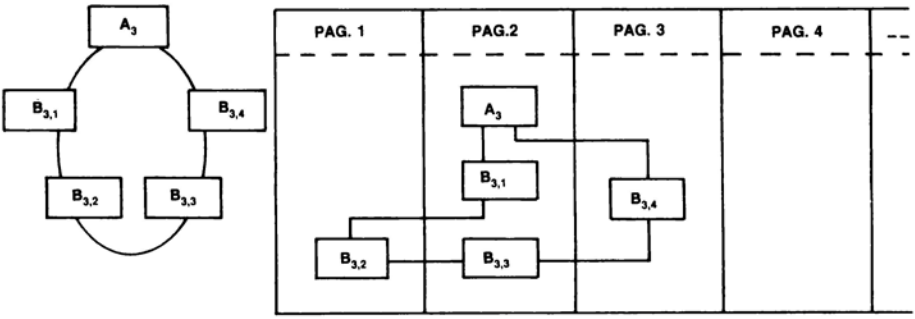


Figura 2.2.e — Disposizione logica e disposizione fisica. Prendendo in esame la catena singola di owner A_3 della figura 2.2.b si nota dallo shorthand, come la sequenza logica sia: A_3, $B_{3,1}$ $B_{3,2}$ $B_{3,4}$. La sequenza fisica potrebbe, ad esempio, essere invece quella del disegno a lato in cui è evidenziata anche la suddivisione in pagine: $B_{3,2}$, A_3, $B_{3,1}$, $B_{3,3}$, $B_{3,4}$. Una situazione di questo tipo si presenta spesso nei casi reali, in funzione dei criteri di caricamento (ed in specie se il Data Base ha subito frequenti aggiornamenti).

A questo punto si potrebbe aprire un lungo discorso su come quell'ordine fisico influenzi le applicazioni, sul fatto se sia possibile ed in quali casi intervenire per stabili-

re quell'ordine (da parte dell'utente) e via dicendo. Questi sono problemi di ottimizzazione assai legati al particolare tipo di sistema. Per essere compresi a fondo, inoltre, richiedono alcuni concetti che affronteremo solo tra varie pagine. Vi torneremo, dunque, al momento opportuno.

2.3 Struttura gerarchica

Ora che si è brevemente visto come viene organizzata l'informazione nei record del Data Base Integrato, è bene vedere quali problemi tale organizzazione permette di risolvere. Nel paragrafo 1.3 si era parlato di strutture logiche di base, di quelle che furono alla nascita dei concetti di integrazione e che condussero ai vari modelli. Si è già detto che tutte quelle strutture, con la sola esclusione della tabellare, sono incluse nelle possibilità del Data Base Integrato: si tratta ora di vedere in che senso.

Ancora si parlerà di *strutture*, sottintendendo, d'ora in avanti, *specifiche* dell'Integrato.

L'analisi che ne faremo ha un duplice scopo: vedere, appunto, come è fatta ogni struttura, quali problemi presenta, quali sono i vantaggi e svantaggi d'uso, ma anche, e soprattutto, studiare il campo d'applicabilità, intuire le possibilità solutive relativamente alle più svariate esigenze logiche che le questioni reali propongono. Non si dimentichi che il Data Base sarà composto da una combinazione più o meno estesa delle strutture che stiamo per esaminare una per una e che alla stesura del disegno complessivo concorreranno vari fattori. Fra questi sono essenziali quello *informativo* (di quali informazioni si ha bisogno) e quello delle *richieste di procedura* (cosa si deve ottenere). Si potrebbe dire che è soprattutto in funzione di questa coppia di fattori che si dovrà decidere struttura per struttura fino a giungere al disegno conclusivo.

Per quanto riguarda, comunque, i problemi di disegno, rimandiamo ai paragrafi relativi: in essi le questioni or ora accennate verranno approfondite meglio. Per ora ci accontenteremo di vedere queste strutture come se dovessero vivere le une separate dalle altre: ciò ci faciliterà nel compito, essenziale, di generalizzare gli esempi sui quali ci baseremo. Essi non sono che casi particolari che ci mostrano la altrettanto particolare capacità solutiva della singola struttura: dovrà inevitabilmente venir fatto uno sforzo per intuire le analogie con altri casi, con tutti quelli che ci si possono presentare.

La prima, più semplice struttura è quella **gerarchica**: l'abbiamo già vista per studiare il set nel paragrafo precedente. Nel notare che essa realizza in modo semplice la relazione uno a molti, se ne era mostrato lo shorthand (vedi disegno 2.2.a) ed un possibile longhand (vedi figura 2.2.b). Vediamone ora alcuni esempi.

Un problema tipico è quello del CLIENTE—ORDINE (tra l'altro di questo esempio si è già fatto uso nelle pagine scorse): come si può osservare in figura 2.3.a, la struttura gerarchica più semplice, ad un solo livello, permette di memorizzare agevolmente tutti i clienti della nostra ideale azienda con elencati tutti gli ordini da essi fatti.

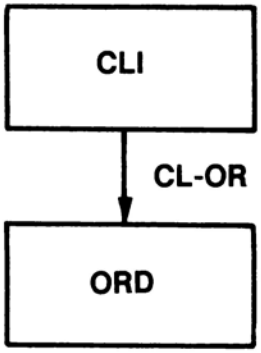


Figura 2.3.a — Struttura gerarchica. Da ogni cliente, memorizzato nel record CLI, vengono fatti dipendere i suoi ordini: i record ORD, member nel set CL-OR.

Non soffermiamoci sui dettagli di questa struttura, non vale la pena, vista la sua estrema semplicità, né esaminarne eventuali longhand, né cercarne più o meno complicate implicazioni logiche. Notiamo, invece, che sicuramente la gerarchia può essere sviluppata su livelli ulteriori. Così, se volessimo memorizzare anche gli articoli che in ogni ordine i vari clienti hanno richiesto basterebbe procedere rendendo il record ORD a sua volta owner in un nuovo set di cui i record articoli (ART) sono i member (vedi figura 2.3.b).

In questo caso soffermiamoci anche ad esaminare un possibile longhand (vedi ancora figura 2.3.b): esso ci servirà ad evidenziare alcune particolarità interessanti.

Prima, tuttavia, notiamo che ancora non è detto che la gerarchia debba fermarsi a questo livello: di nuovo il record ART potrebbe essere owner in un ulteriore set. Di fatto potremmo generare gerarchie a molti livelli, fino ad un massimo che dipende strettamente dal sistema specifico, ma che è sempre dell'ordine di diverse decine.

Consideriamo ancora la figura 2.3.b: dal longhand risultano evidenti alcuni fatti importanti: se desideriamo, dato un cliente, esaminarne tutti gli ordini, la ricerca ri-

sulta piuttosto facile: è sufficiente scorrere la catena singola relativa al cliente desiderato. Se desideriamo, d'altro canto, dato un certo ordine, determinare quale cliente lo ha fatto, la ricerca risulta altrettanto facile: non dimentichiamoci dell'esistenza del pointer owner che ci assicura, una volta determinato il record ORD voluto, di poter risalire direttamente all'appropriato cliente.

Qualche difficoltà potrà esistere nell'accesso all'ordine (alla sua determinazione), tuttavia ciò potrebbe essere a carico del sistema, essendo l'ordine univoco, non confondibile con tutti gli altri (*). Comunque, questo problema della univocità della «voce» cercata implica un accesso più o meno facile e veloce (e ciò indipendentemente dalla disposizione fisica del record nelle aree di registrazione). In altre parole: l'accesso alla *voce* desiderata potrà essere più o meno facile, ma, sicuramente risulterà possibile in un sol colpo, solo nel caso di individuazione su *voce* unica.

Si pensi, infatti, agli articoli: se desiderassimo, dato l'articolo di codice ART6 risalire a quali clienti lo hanno ordinato (si veda sempre la figura 2.3.b) ci troveremmo in qualche difficoltà: quell'articolo è stato ordinato da molti: ciò ci costringe a scorrere tutti i record ART, per ognuno di essi a risalire al suo owner e via dicendo. Fatto sta che dovremmo scorrere uno per uno tutti gli articoli selezionando i desiderati con ovvia, conseguente, perdita di efficienza (e di tempo).

Tutto ciò per non parlare dello spazio: se il record ART è di dimensioni notevoli, il fatto che lo stesso articolo possa comparire più volte (si pensi ad ART2 o ad ART6 nel solito longhand di figura 2.3.b) provoca sprechi eccessivi: meglio sarebbe che ogni articolo comparisse una volta sola, tanto per le possibilità di accesso, quanto per il risparmio di spazio.

Questo tipo di problema non ci deve spaventare: le richieste di procedura (ciò che ci proponiamo di ottenere dalle applicazioni) possono essere tali da farci accettare compiutamente la struttura disegnata, e per la sua semplicità, e per la sua completezza. Il discorso affrontato sulle possibilità di accesso e sulle ridondanze (come

(*) Si potrebbe, in altre parole, utilizzare un campo del record ORD come chiave di accesso.

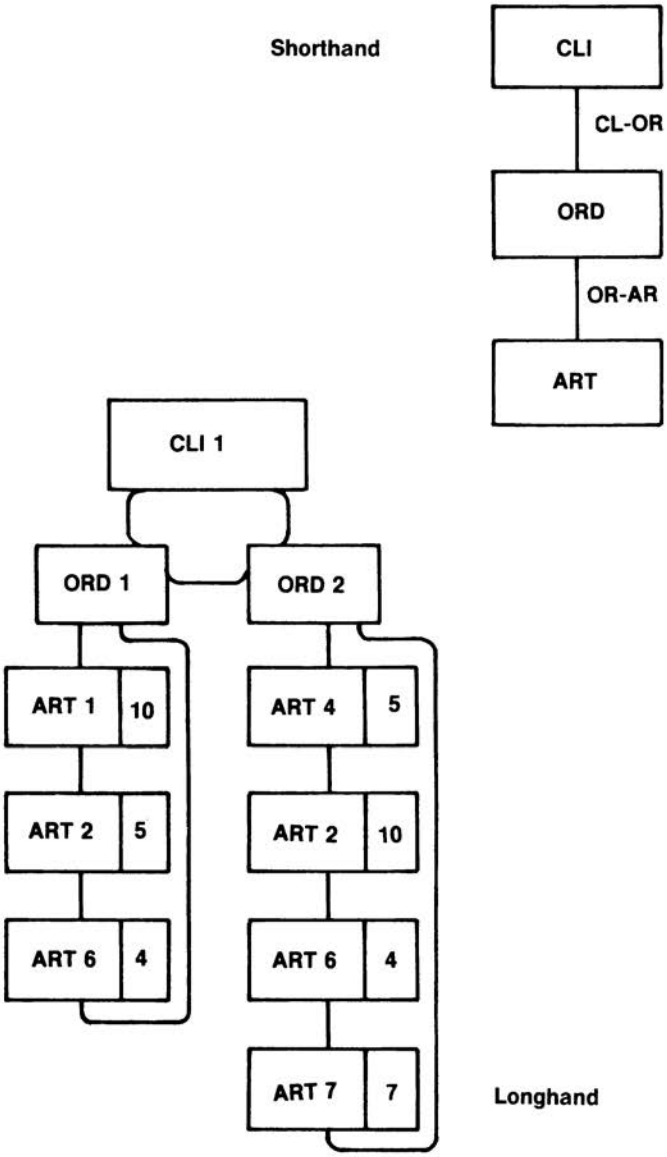
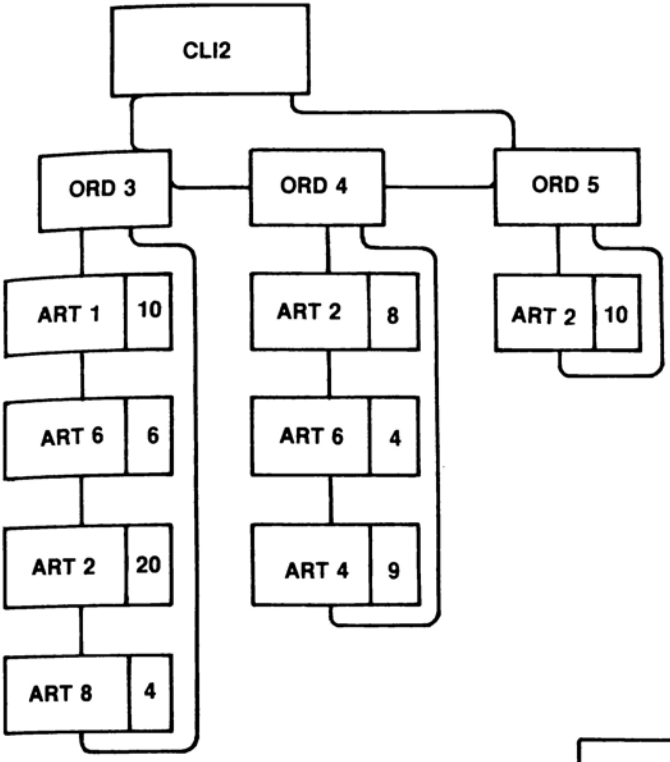
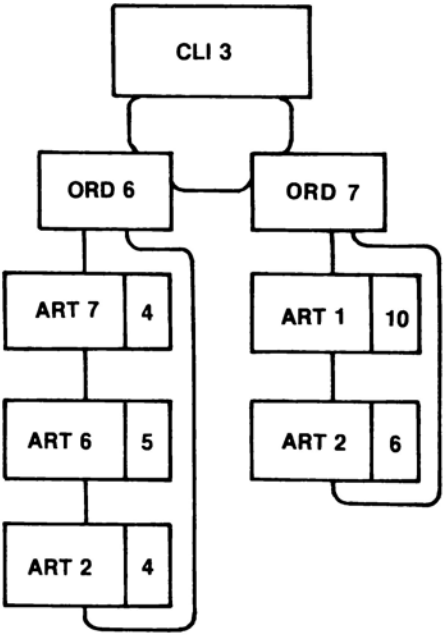


Figura 2.3.b — Struttura gerarchica: shorthand e longhand. Il record ART (come del resto gli altri) può recare in sé varie informazioni, nel longhand sono evidenziati solamente il codice e la quantità dell'articolo ordinato.


CLI2
ORD 3
ORD 4
ORD 5
ART 1
10
ART 6
6
ART 2
20
ART 8
4
ART 2
8
ART 6
4
ART 4
9
ART 2
10


CLI 3
ORD 6
ORD 7
ART 7
4
ART 6
5
ART 2
4
ART 1
10
ART 2
6

quelle di ART2) vuol solo significare che anche in essa, così semplice ed immediata, possono esistere dei problemi e che, qualora quelle richieste di procedura non vogliano altrimenti, può essere preferibile decidere per delle strutture diverse (ne vedremo degli esempi).

Sta di fatto che la relazione uno a molti è quella più frequente nei problemi che la realtà ci propone e quella che, quindi, più facilmente costituirà la base del disegno del nostro Data Base. Essa potrà completarsi con altre relazioni a formare strutture più o meno complicate, sicuramente, però, sarà il punto di partenza di ogni problema applicativo, considerata, come si è già detto, la sua semplicità logica, nonché la sua frequenza nel rispecchiare problemi reali.

Altri esempi, che ci torneranno utili anche più oltre, possono essere quelli di figura 2.3.c e 2.3.d: nel primo la nostra ideale azienda è stata suddivisa in settori, ogni settore in uffici in ognuno dei quali lavorano dei dipendenti. Nel secondo si è immaginato di voler memorizzare gli studenti di una certa università nonché la facoltà di appartenenza, gli esami sostenuti e le votazioni.

Fissiamo le idee sul caso della figura 2.3.c: può accadere che in certi uffici venga assunto, ogni tanto, del personale avventizio, a tempo determinato; immaginiamo di

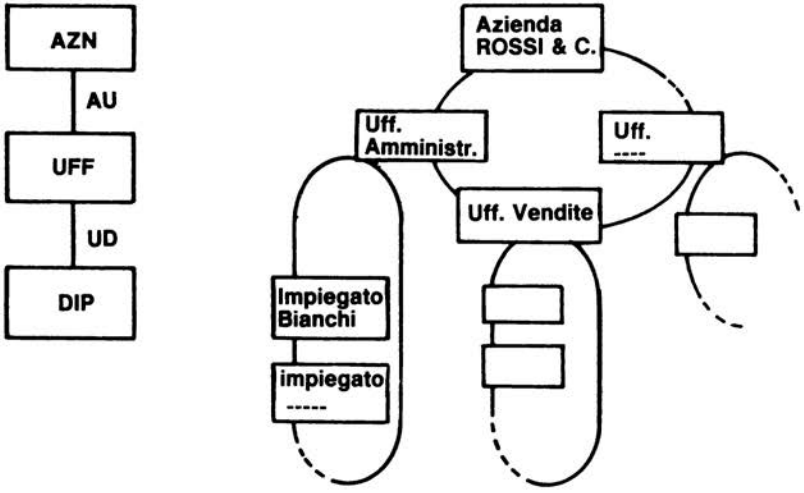


Figura 2.3.c — Un altro caso di struttura gerarchica su più livelli: la presenza dell'unico record AZN ci assicura una *entry* univoca, da cui tutte le altre informazioni dipendono.

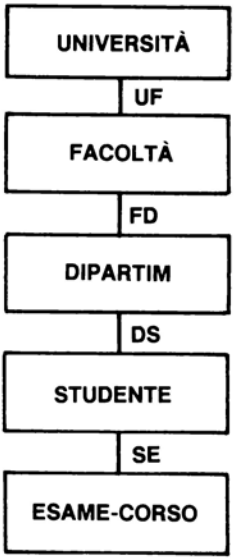


Figura 2.3.d — La struttura gerarchica può estendersi su un alto numero di livelli: ogni DBMS che la permette ha un suo proprio limite massimo per il numero di record in gerarchia che è quasi sempre dell'ordine di alcune decine.

voler rendere riconoscibile questa situazione. La prima cosa che viene in mente è di porre sul record DIP una indicazione sul tipo di dipendente, ma questo è qualcosa in più, un codice che siamo stati costretti ad inserire ad hoc, che ha ingrandito, magari anche di poco, il record: sarebbe sicuramente preferibile disporre di due record diversi, ad esempio DIP-I per il personale a tempo indeterminato e DIP-D per quello a tempo determinato. La struttura gerarchica permette anche questo, in quanto non esiste alcun limite a porre nello stesso set qualsivoglia numero di tipi record diversi (si osservi la figura 2.3.e).

Come si osserverà dal longhand, nelle catene singole, degli uffici in cui è presente personale di entrambi i tipi, i record diversi risultano mescolati liberamente (*): sarà possibile procedere ad una selezione e sarà sempre possibile eseguire uno scorrimento ad esempio saltando quelli del tipo non desiderato.

Non è detto che in un set possano trovarsi solo due diversi tipi di record, si è già notato che non esiste un lioite teorico a questo numero. Osserviamo la figura 2.3.f.

(*) In alcuni sistemi, nella catena singola si trovano prima tutti i record di un tipo, poi quelli dell'altro.

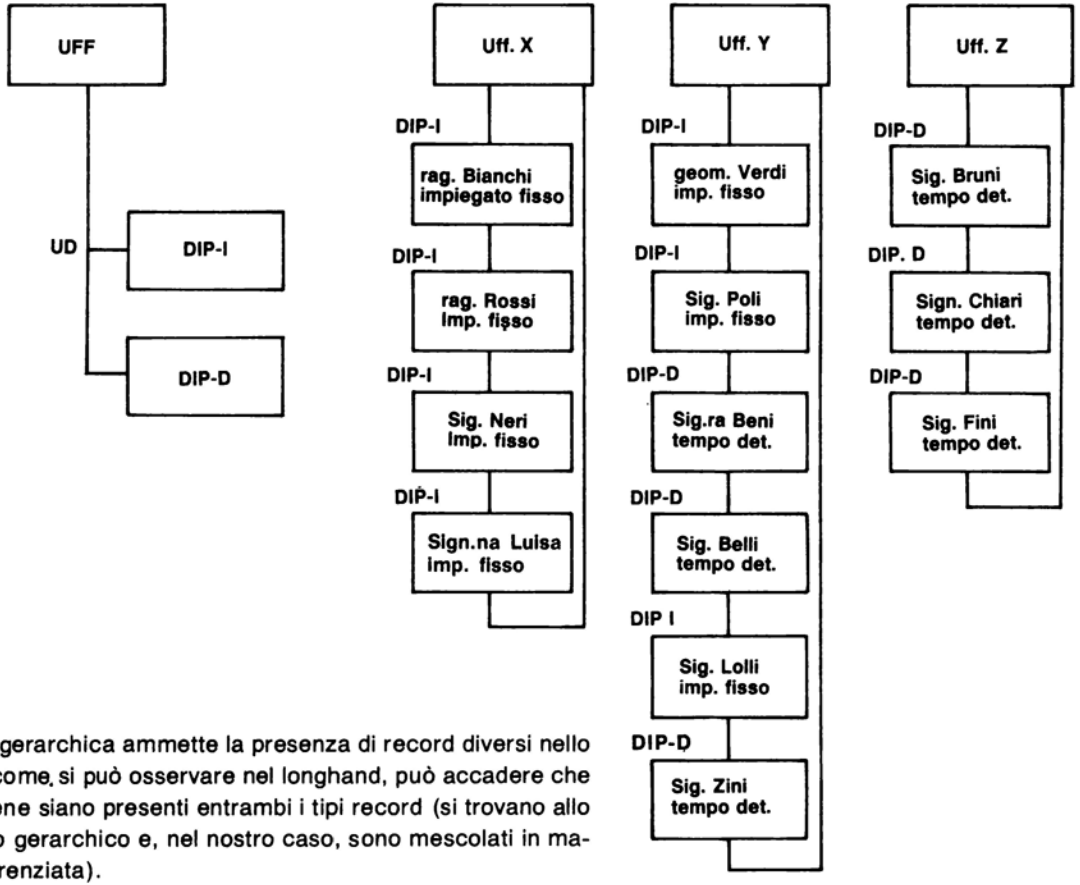


Figura 2.3.e — La struttura gerarchica ammette la presenza di record diversi nello stesso set: come si può osservare nel longhand, può accadere che in certe catene siano presenti entrambi i tipi record (si trovano allo stesso livello gerarchico e, nel nostro caso, sono mescolati in maniera indifferenziata).

Si è immaginata un'azienda assicuratrice che propone ai suoi clienti quattro tipi diversi di assistenza: polizza auto, casa, medica e vita. Si è anche immaginato di voler memorizzare i clienti che hanno quelle polizze (uno stesso cliente può avere più polizze e quindi comparire in più set).

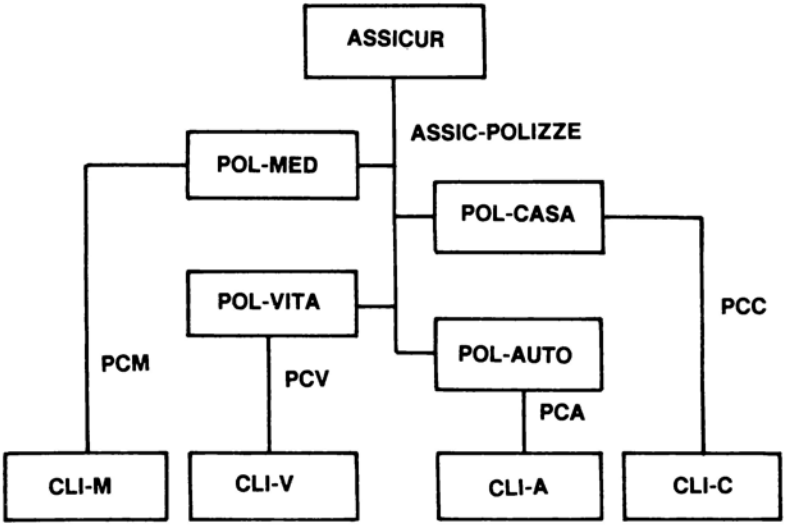


Figura 2.3.f — Estensione della struttura gerarchica con tipi record diversi nello stesso set, ad un ulteriore livello di dipendenza (i record relativi ai clienti). Certi clienti potrebbero risultare duplicati (quelli che hanno più polizze) né più né meno come nel caso della struttura ordine-articolo.

Come è facile osservare la nostra struttura gerarchica è cresciuta e sta diventando forse un po' composita. Anche questo non ci deve spaventare: soluzioni di questo tipo sono in realtà assai semplici da gestire e possono rispondere appieno alle richieste di procedura. Forse lo stesso problema, ci si obietterà, potrebbe essere risolto in altro modo, magari più semplice (ad esempio evitando di duplicare i clienti). La risposta non potrebbe che essere affermativa, tuttavia entreremo nel merito del discorso solo più oltre.

Restando invece in discorsi di carattere puramente strutturale, notiamo che il caso visto può corrispondere anche ad una struttura ad albero. In realtà, da un punto di vista logico, abbiamo proprio generato un albero, tuttavia, è solo una questione di terminologia, abbiamo preferito battezzare con questo nome una struttura leggermente diversa: la analizzeremo nel prossimo paragrafo.

2.4 Struttura ad albero

Se la struttura del set con più member di tipo diverso è, a ben vedere, una gerarchia a tutti gli effetti (ed è per questo che l'abbiamo esaminata nel paragrafo precedente sotto quel titolo), essa però risolve spesso anche quei problemi in cui la tipica struttura ad albero sembra la più aderente. Lo si era già notato e non si può che ribadire che, ad esempio, casi come quelli della figura 2.3.e e 2.3.f sono comunque strutture ad albero (*la relazione uno-molti, sviluppandosi su vari livelli, moltiplica il numero del tipo diverso di informazioni*).

È dunque più per motivi di terminologia e di facilità di distinzione che battezzeremo con **struttura ad albero** un nuovo tipo di struttura base: quello in cui un owner è tale, relativamente a record diversi, in set diversi.

Il caso più semplice è quello di figura 2.4.a: come si noterà è ancora l'esempio dei dipendenti assunti a tempo indeterminato o a tempo determinato della figura 2.3.c, ma ora si è scelto di porre le due categorie di dipendenti su due set completamente diversi. Se si confronta il longhand di figura 2.4.a con quello della figura 2.3.c le poche semplici differenze concettuali, ma anche di struttura, balzano agli occhi.

Si noterà ad esempio come il set UDD presenti *catena vuota* per l'ufficio X, viceversa per l'ufficio Z è UDI ad essere vuoto. Si noterà anche come ogni record UFF questa volta disponga comunque di quattro pointer, nel caso della figura 2.3.c esso dispone di due soli pointer (si ricordi quanto detto nel paragrafo 2.2 e si riveda la figura 2.2.d).

Non andiamo a cercare altre particolarità o caratteristiche strane che, peraltro, questa semplice struttura non presenta; caso mai può essere interessante porsi la domanda: quando è preferibile la struttura ad albero nei confronti di quella gerarchica con più tipi record nel set, e quando il viceversa? Ancora le risposte dovranno conseguire da una *regola* di comportamento ben precisa, già notata nel paragrafo precedente: *sono comunque le richieste di procedura, o applicative, che ci devono guidare nella scelta della struttura più opportuna*. Così, confrontiamo ancora le figure 2.3.c e 2.4.a, se esistesse la necessità di scorrere i set, in maniera frequente, selezionando uno dei due tipi record, la presenza anche dell'altro potrebbe provocare qualche perdita di tempo.

Precisiamo con un esempio: prima di tutto ammettiamo che nell'ufficio Y, che nella prima delle due figure in questione ha la catena conclusa da un tratteggio poiché ha un numero imprecisato di dipendenti, abbia in realtà un numero *alto* di dipendenti. Ammettiamo anche che una delle richieste applicative sia quella di poter accedere ad un qualsiasi ufficio (come il nostro Y) e di ottenere l'elenco di tutti i dipendenti di tipo DIP-I o DIP-D indifferentemente, ma solo quelli. Una volta che sia previsto un qualunque metodo di accesso al record UFF, la ricerca può procedere per scorrimento della catena (ovvero mediante lettura dei pointer) accedendo ad ogni member del tipo voluto. È ovvio che la presenza di record di tipo diverso (il primo caso, ovvero quello di figura 2.3.e) costringe il sistema a leggere i pointer anche dei record che non interessano e dunque a fare degli accessi in più.

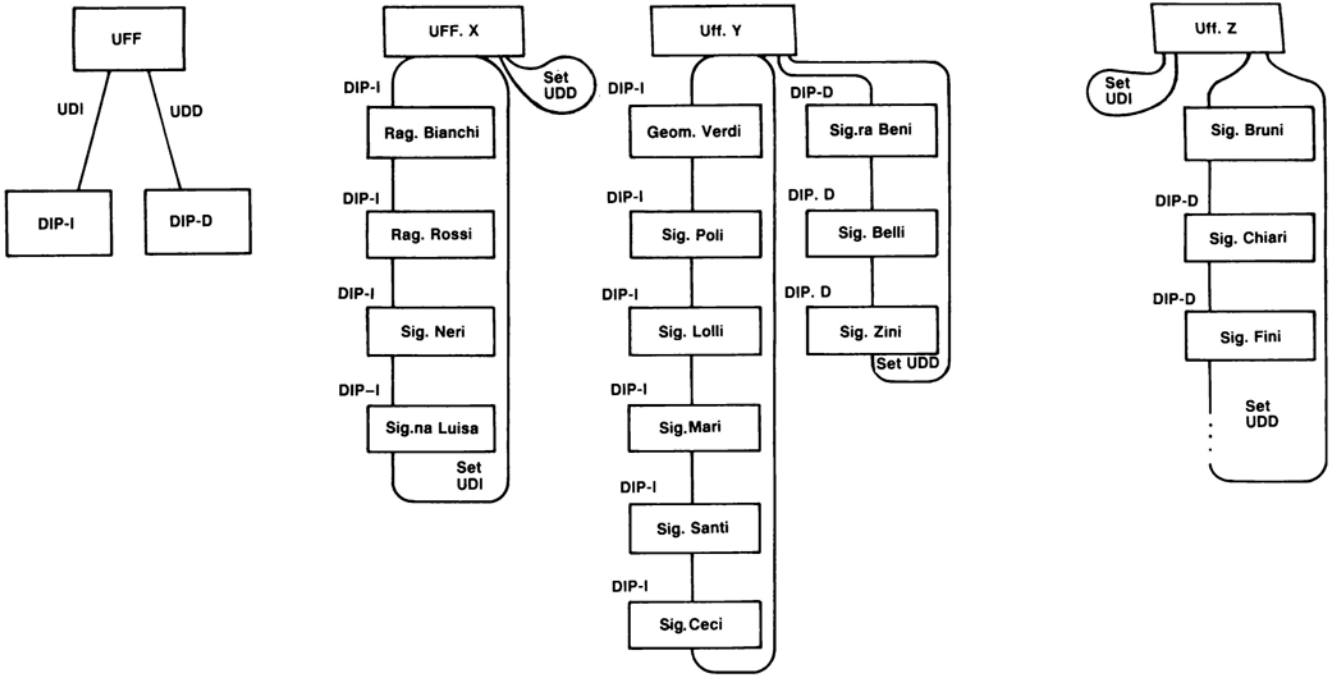


Figura 2.4.a — La struttura ad albero: un owner è tale rispetto member diversi in set diversi. In questo caso la struttura è assai semplice (anzi, la più semplice), ma si potrebbe avere un più alto numero di set e un maggiore numero di livelli (vedi prossimi esempi) realizzando il vero e proprio *grafo* dell'albero. Si noti che si tratta di qualcosa di analogo all'esempio di figura 2.3.e (l'ufficio Y è stato, però, completato: esso contiene sei record di tipo DIP-I, a differenza di quanto avveniva nell'esempio precedente, ciò per dare un'idea di sia pur parziale *numerosità*).

Per scendere ancor più in dettaglio si osservi la figura 2.4.b, in essa si è disegnato quella che potrebbe essere la distribuzione fisica dei record DIP-I e DIP-D nelle varie pagine dell'archivio sia nel caso di un unico set come in figura 2.3.e, sia in quello di una struttura come quella di figura 2.4.a. Forse va ribadito: *potrebbe essere*! Nulla in effetti impedisce di generare pagine più grandi, così grandi, magari, da far stare tutto quanto l'ufficio X con tutti i suoi dipendenti in una sola pagina. In realtà questo non accade praticamente mai per quei set che hanno un numero elevato di members: grosse dimensioni per la pagina fisica risultano, alla fine sconsigliabili. Il problema è quello dei *raggruppamenti* delle informazioni e più in generale delle ottimizzazioni così ottenibili. Poichè il discorso non può che essere legato al particolare DBMS in questione, non lo affronteremo in questa sede. Invece fissiamo le idee sul fatto che le situazioni come quella della figura 2.4.b possono essere dovute a come è stato eseguito il caricamento, magari, a certi criteri di allocazione dei records, spesso al fatto che vi sia stato un certo numero di aggiornamenti. Si prenda comunque come dato di fatto che le due situazioni non sono straordinarie, che, anzi, esse rappresentano la normalità (in senso statistico) delle situazioni.

Fatta tale ipotesi si consideri lo scorrimento di UD, a partire dall'owner Y, alla ricerca di tutti i record DIP-I: dopo un primo accesso in pagina 20, che ci assicura la disponibilità di Y, di VERDI e di POLI (*) dovremo eseguirne un secondo (inutile...) per *attraversare* BENI e BELLI, un terzo accesso ci permetterà di raggiungere LOLLI, un quarto (inutile...) ci farà *attraversare* ZINI per giungere finalmente con un quinto, a MARI, SANTI, e CECI.

Se avessimo dovuto ricercare i DIP-D non ci sarebbe andata meglio: dopo un primo accesso ad Y, attraversando VERDI e POLI, un secondo accesso ci avrebbe messo a disposizione LOLLI e così via, come prima. Prima di vedere terminata la selezione di tutti i record voluti, avremmo ancora dovuto eseguire, appunto come prima, un totale di cinque accessi.

Con due set diversi, appunto UDI e UDD, le cose sarebbero andate meglio?

Vediamo: la ricerca dei DIP-I corrisponde allo scorrimento di UDI, mentre quella dei DIP-D allo scorrimento di UDD. Scorrendo UDI dopo un primo accesso in pagina 20, dobbiamo farne un secondo per trovare LOLLI ed infine, con un terzo in pagina 23, abbiamo concluso. Se poi dovessimo cercare i record DIP-D, basterebbero come è ormai ovvio due soli accessi. Le cose dunque, in questo secondo caso sembrano andare decisamente meglio: o due, o tre accessi, contro, comunque, cinque.

La realtà è che, inutile dirlo, è la richiesta di procura ed essa sola che può farci decidere quale delle due situazioni sia la migliore. In effetti se essa chiede l'accesso a tutti i dipendenti, le due situazioni sono indifferenti come numero d'accessi e dun-

---

(*) Accesso fisico è l'atto di portare nel buffer di memoria una intera pagina: a questo punto tutto il contenuto di pagina è a disposizione dell'elaborazione; la maggior perdita di tempo, per il sistema, è dovuta all'accesso fisico — operazione di I/0 — e non certo all'elaborazione, di qui la necessità di minimizzarne il numero, ossia, in definitiva, di cambiar pagina il meno possibile.

que altrettanto adatte ad un'elaborazione veloce; la prima in tal caso presenta il vantaggio di risparmiare un poco di spazio: 2 pointers contro 4 per ogni owner. Su un numero basso di record questo non è un risparmio apprezzabile; d'altra parte bisogna anche pensare che sui grandi numeri il badare ad una certa economia nelle occupazioni può risultare indispensabile: ad esempio già con una presenza di 1000 owner il primo caso fa perdere 20.000 pointers, il secondo 40.000 (con una differenza di 20.000!).

Se, invece, è più frequente dover accedere ad uno dei due tipi di record e ad esso soltanto, qualunque esso sia, la soluzione dei set separati sembra vantaggiosa da un punto di vista della velocità di elaborazione (e di solito ciò risulta più importante di qualunque altra preoccupazione).

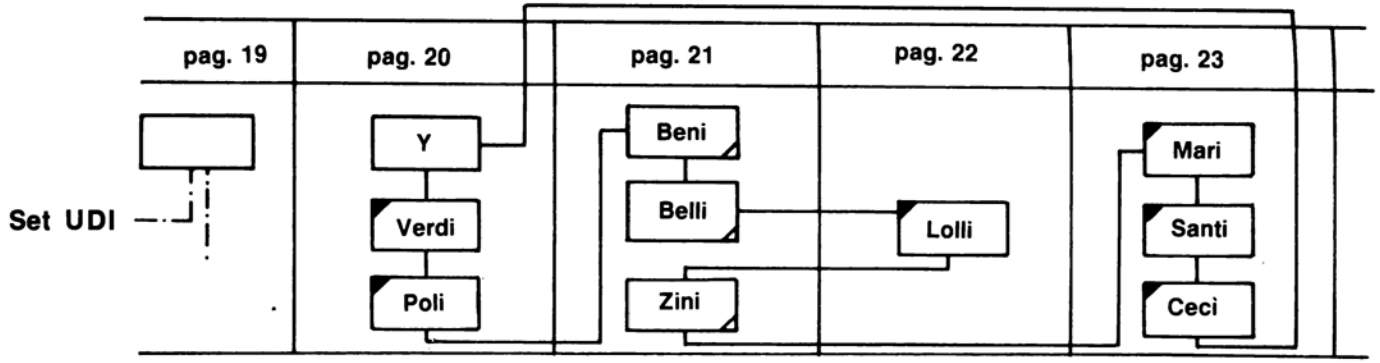
Da questo discorso non va tratta altra conclusione se non che la famosa regola delle richieste di procedura non può che risultare decisiva.

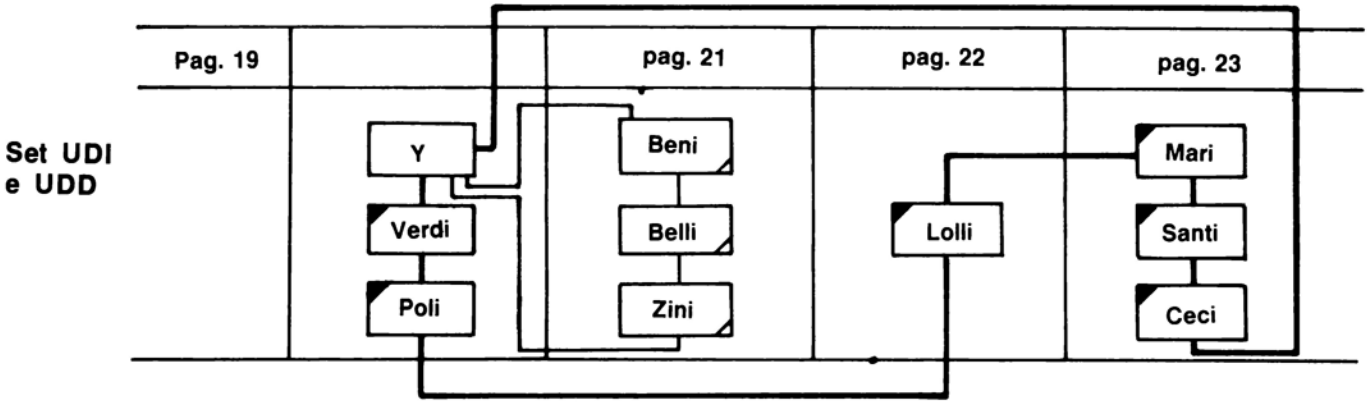
Indipendentemente da tutto ciò la struttura ad albero in sé presenta un qualche interesse non ancora del tutto approfondito dal punto di vista logico: quando si dice *grafo ad albero* si intende, generalmente, la realizzazione grafica di una relazione uno-molti che si ripete più volte su livelli gerarchici sempre più bassi. Si osservi, allora, la figura 2.4.c., in essa è evidenziato un grafo ad albero: da A (elemento qualunque) dipendono gli altri elementi (altrettanto qualunque) B1, B2,...., Bn, da ognuno di questi dipendono, a loro volta gli elementi C11, C12,...C1h, C21, C22,...,C2k, ...ecc.

Il significato della parola *albero* sta proprio nella realizzazione grafica di questo concetto, si pensi soltanto al senso logico che si dà comunemente all'*albero genealogico* in termini (più o meno ornamentali...) di struttura.

Con tutto ciò non si intendeva che suggerire dei concetti di relazione (come ormai sarà ben chiaro) per poterli estendere a casi di un certo interesse pratico: ad esempio considerando sempre la stessa figura ci si renderà conto che se ogni owner, o per trasporre il termine al grafo, *padre*, genera un numero non troppo limitato di *figli* o member, l'albero vero e proprio può essere proprio il più conveniente. Quanto meno lo è qualora i diversi *figli* siano tra loro, da un punto di vista logico od applicativo, completamente indipendenti. Altrimenti la soluzione di porre tutti i *figli* nello stesso set può ancora essere la più vantaggiosa. A rischio di ripetersi in modo fastidioso: le richieste di procedura e la *fantasia* del *disegnatore* non possono che indirizzare verso la strada più opportuna; l'*invenzione* della migliore delle strutture.

Figura 2.4.b — La disposizione fisica dei record nei casi di un solo set oppure di due diversi: si tratta, ovviamente di *una* delle *possibili* disposizioni, tuttavia può essere assunta quale situazione *tipo*. Ovviamente il numero di record member è comunque limitato: si deve immaginare di estendere la struttura e tutto il disegno, che non cambiano caratteristiche, anche a numeri assai più elevati (il numero di accessi aumenterà di conseguenza). I record di tipo DIP-I da un triangolo scuro in alto a sinistra, quelli di tipo DIP-D da un triangolo chiaro in basso a destra; nel caso dei due set differenti il fatto che una delle due catene sia disegnata in maniera più marcata serve solo ad un più facile riconoscimento visivo e non ha alcun senso elettivo o di privilegio nei confronti dell'altra.


pag. 19
pag. 20
pag. 21
pag. 22
pag. 23
Set UDI
Y
Verdi
Poli
Beni
Belli
Zini
Lolli
Mari
Santi
Ceci


Pag. 19
pag. 21
pag. 22
pag. 23
Set UDI
e UDD
Y
Verdi
Poli
Beni
Belli
Zini
Lolli
Mari
Santi
Ceci

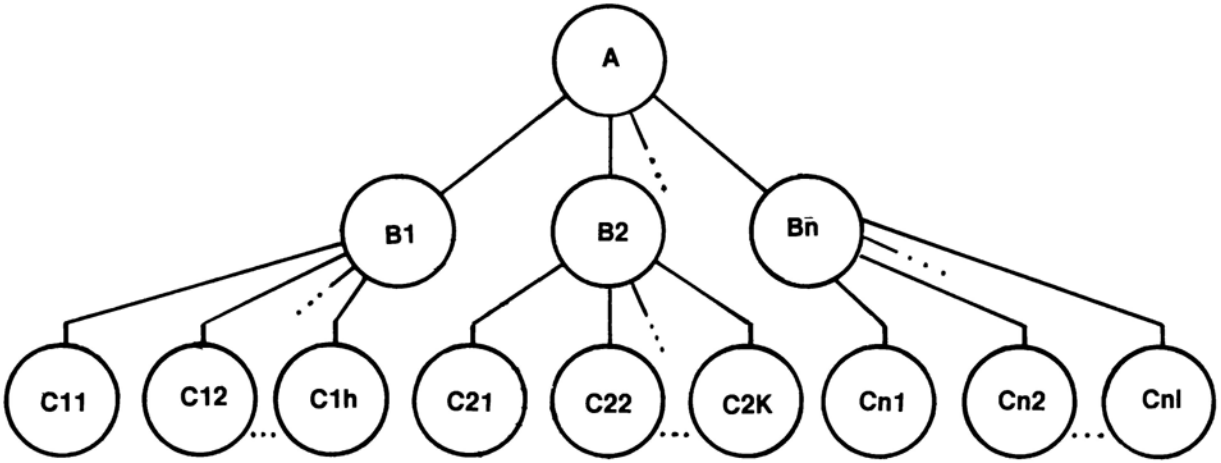


Figura 2.4.c — La struttura ad albero (come già vista nel paragrafo 1.5) può essere intesa come *modello logico di base*. La sua realizzazione nell'Integrato risulta agevole attraverso l'istituzione di set diversi; essa risulta (sempre nell'Integrato) fondamentale nel rappresentare la dipendenza dei *figli* dal *padre* e, contemporaneamente la completa indipendenza reciproca dei *figli*.

2.5 Strutture reticolari: il record connettore

Normalmente si dice che il Data Base Integrato si fonda sul modello reticolare. Ciò è dovuto al fatto che esso permette l'uso, tra le altre, di alcune particolari strutture che prendono nome di **reticolari**. La loro presenza assicura (si ricordi quanto osservato nel paragrafo 1.6) un'ampia possibilità associativa delle informazioni: nel modello reticolare, si era detto, ogni entità (o record) può essere vista come un *nodo* di più relazioni... Insomma possiamo dire che si può parlare di Data Base reticolare proprio per generalizzazione del nome di alcune sue strutture particolari che ne rappresentano e sottolineano l'importanza.

Quanto esse siano importanti lo si può facilmente intuire una volta tenuto conto delle possibilità solutive che offrono, queste a loro volta, risulteranno evidenti dall'analisi e dagli esempi che seguono (non si dimentichi che, come già si è detto, sarà la combinazione di tutte le strutture base a fornire la struttura generale del Data Base, solutiva del problema applicativo).

La prima delle strutture reticolari che vogliamo esaminare è quella con *record connettore*, o, più brevemente, *reticolare a connettore*. Essa è schematizzata in figura 2.5.a e, come il disegno stesso suggerisce, ha l'importante funzione di connettere (da cui il nome) due record di tipo diverso, owner in due set diversi di un member comune, il **record connettore** vero e proprio.

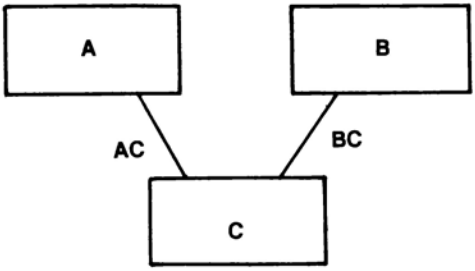


Figura 2.5.a — Struttura reticolare con record connettore. Dati n record A ed m record B (con $n \gtreqless m$) i vari C hanno la funzione di mettere in relazione, o connettere, un particolare A con un particolare B. Si può anche dire che ogni connettore individua una particolare coppia di owner. Il record C può contenere, o meno, delle informazioni logiche.

L'aver stabilito una *connessione* non è altro che l'aver trasposto una relazione tra un certo A ed un certo B, senza che a priori vi fosse una precisa corrispondenza numerica tra i due tipi record. In effetti si può pensare di avere n record A ed m record B dove m n. Insomma non si tratta più della realizzazione di una relazione uno-molti, per la quale, come abbiamo visto, la struttura gerarchica risulta essere la più aderente alla logica del problema, bensì di una relazione da stabilirsi tra certi singoli elementi la cui totalità è però a priori indipendente, come indipendente è l'informazione logica che essi rappresentano.

Si osservi la figura 2.5.b: si tratta di un possibile longhand della struttura in esame. Come si noterà la funzione dei connettori (tipo record C) è quella di definire le varie coppie A_1B_1, A_1B_2, A_1B_4, A_2B_1, A_2B_2, A_3B_1, A_3B_2. Non si tratta ovviamente di tutte le coppie possibili, ma solo di quelle che ci possono interessare per un qualche motivo; questo non è che un esempio del tutto teorico, ma il concetto di base è: *dovremo inserire un record C ed uno solo per ognuna delle coppie desiderate*. Il totale di tutte le coppie sarebbe stato 12 nel nostro caso specifico. Più in generale il massimo numero di record connettori per n record A e per m record B è $m \times n$.

Il discorso fatto vale ovviamente nella sola ipotesi che abbiamo scelto: di individuare con i connettori delle *coppie* in quanto tali, ma nulla avrebbe impedito l'inserimento di un più alto numero di connettori.

Ad esempio in figura 2.5.c è ancora presente lo stesso longhand della figura 2.5.b e ancora sono le stesse le coppie individuate. Tuttavia la coppia A_3B_2 è ora definita da due connettori C_7 e C_8; ripetiamo: ciò non è né impossibile, né assurdo e neppure, come potrebbe semmai sembrare, inutile. Dipende semplicemente dal caso specifico.

A questo proposito, infatti, vanno fatte alcune osservazioni sui contenuti dello stesso record connettore. Nell'ipotesi che esso debba servire esclusivamente a stabilire delle relazioni, ovvero definire la coppia, non è ovviamente necessario che esso abbia un particolare contenuto informativo: si parla allora di *connettore vuoto* ed il record è costituito esclusivamente dal pointer. Non occorre precisare che allora la duplicazione di connettori, come accade per C_7 e C_8, che definiscono la stessa coppia, è del tutto inutile e da evitare per non generare confusione.

D'altra parte può accadere che il record connettore debba avere un suo contenuto. In tal caso esso *qualifica* la coppia, definisce un *attributo* per la relazione e può dunque rendersi utile o addirittura necessaria la duplicazione: ad esempio potrebbe accadere, sempre riferendosi alla figura 2.5.c, che C_7 e C_8 definiscano due attributi diversi della stessa coppia A_3B_2.

Affrontiamo, in proposito, qualche esempio maggiormente concreto. Un caso abbastanza tipico: supponiamo di aver a che fare con un'azienda che offre una serie di servizi ai propri clienti, supponiamo anche di sapere con certezza che ogni cliente usufruisce di un unico servizio. Il problema di memorizzare la situazione è risolto in modo assai semplice dalla struttura a connettore come in figura 2.5.d: saranno sufficienti, ovviamente, dei connettori vuoti.

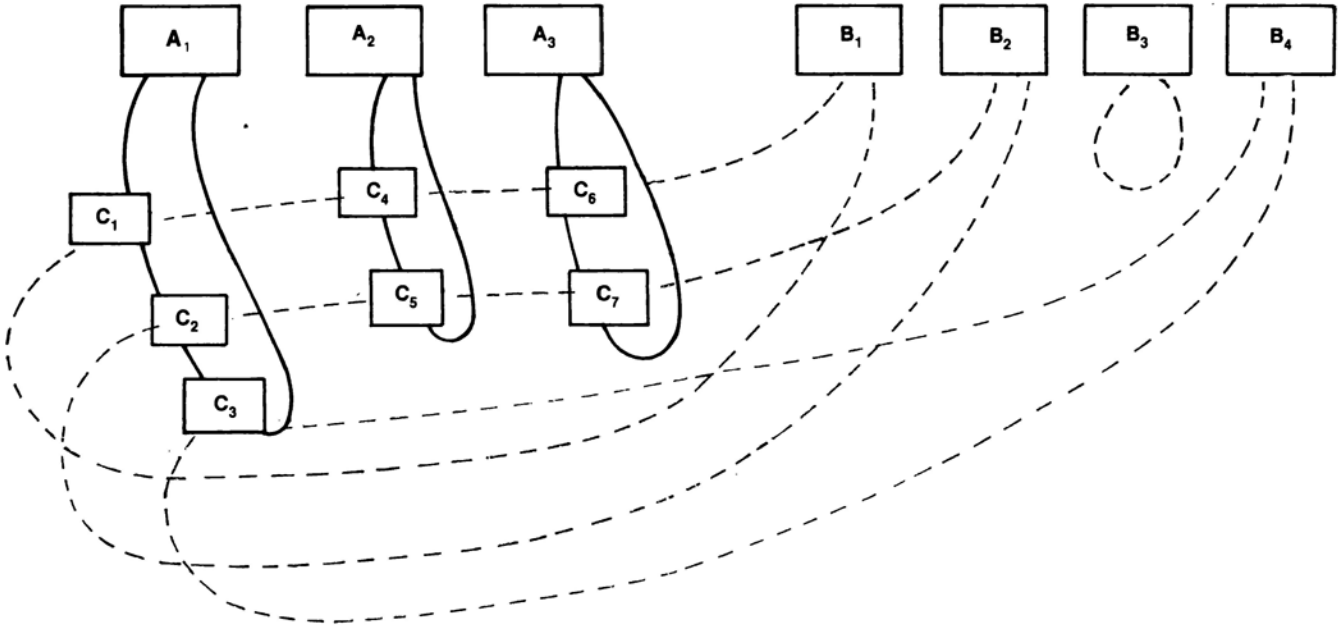


Figura 2.5.b — Longhand: un possibile esempio relativo alla struttura della figura precedente. I due set sono disegnati in colori diversi (AC in nero, BC tratteggiata in rosso) unicamente per facilitarne il riconoscimento.

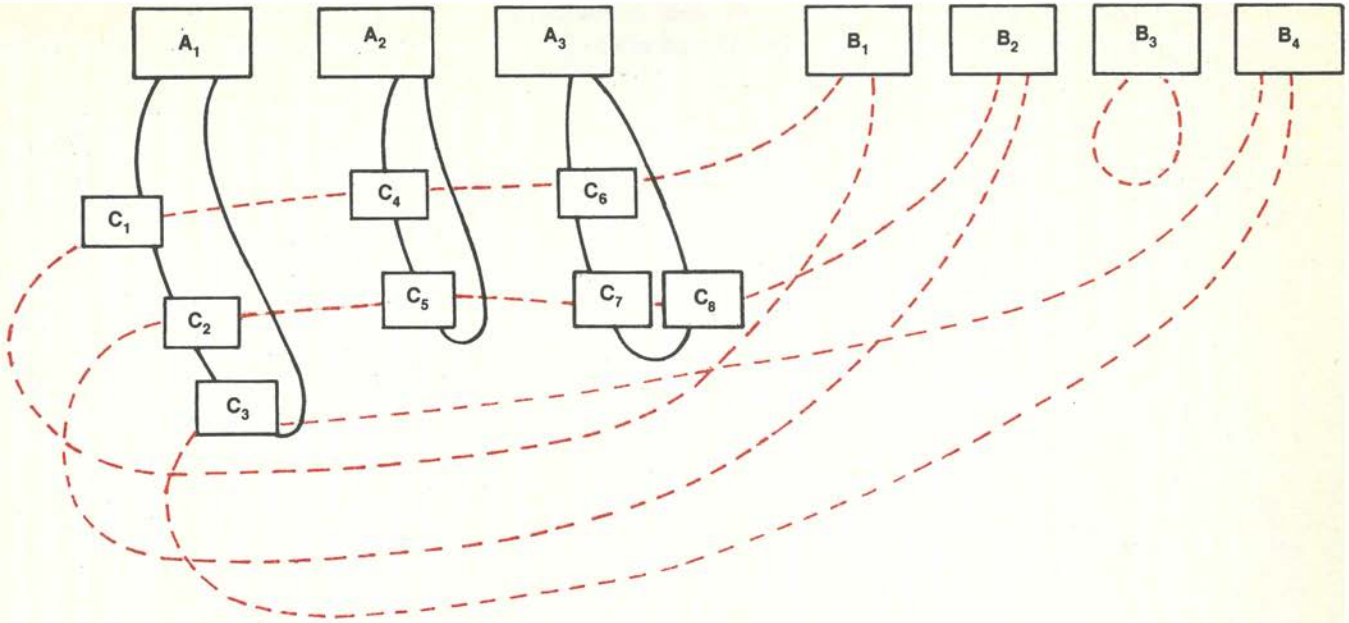


Figura 2.5.c — È ancora un possibile longhand della struttura di figura 2.5.a., questa volta, però, la coppia A_3B_2 è individuata sia da C_7 che da C_8. Apparentemente la presenza anche di C_8 è del tutto inutile, tuttavia, in alcuni casi, può accadere che la coppia sia individuata da *attributi* diversi che allora possono essere inseriti come, appunto, contenuti diversi di due connettori come C_7 e C_8.

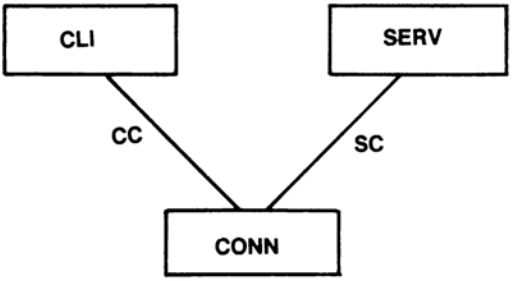


Figura 2.5.d — Il connettore definisce una coppia CLIENTE-SERVIZIO. Esso ha, in questo caso, tale unica funzione ed è dunque privo di informazione *logica*.

Proprio questo è dunque un caso in cui il connettore (*vuoto*, che, cioè, non qualifica, o non definisce attributi) individua delle coppie. Supponiamo che i clienti possano usufruire di più servizi: dovremo inserire un maggior numero di connettori, ma la struttura non cambia. Al proposito si mettano a confronto i due longhand di figura 2.5.e entrambi relativi alla stessa struttura, quella di figura 2.5.d: nel primo caso i clienti usufruiscono di un unico servizio, nel secondo di quanti si voglia servizi.

Nel caso in cui i connettori definiscano un attributo di coppia, le cose possono essere leggermente diverse, anche se il concetto di base non cambia. A questo proposito si torni al problema CLIENTE-ORDINE-ARTICOLO che abbiamo esaminato nel paragrafo 2.3. Si è notato allora che la struttura gerarchica, (esaminiamo nuovamente la figura 2.3.b) risolve sì il problema, ma a prezzo di una duplicazione degli articoli. Ciò può comportare a propria volta difficoltà d'accesso e di ricerca nella fase applicativa. Con l'aiuto del record connettore siamo in grado di evitare qualunque ridondanza: porremo gli articoli e i clienti in record non gerarchicamente dipendenti tra loro, e dunque non duplicati, ed utilizzeremo la *quantità ordinata* come connettore (ovvero come attributo di connessione) a definire *quanti di quegli articoli sono stati ordinati in quell'ordine* (la presenza del record cliente, per quanto riguarda il problema che stiamo esaminando, è del tutto inessenziale). Per verificare quanto detto è sufficiente un paragone attento delle figure 2.3.b e 2.5.f: quest'ultima presenta la soluzione con connettore e si spiega da sola.

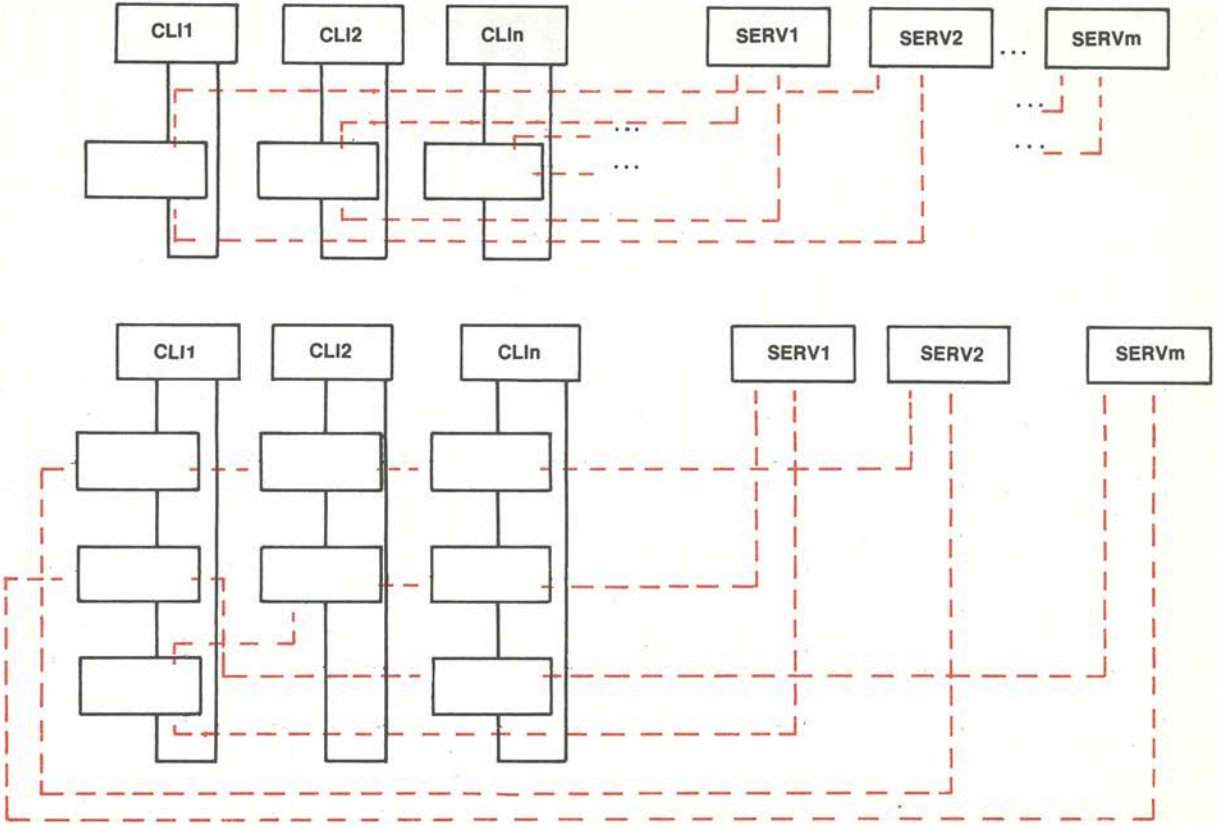


Figura 2.5.e — Due tra i possibili longhand della struttura CLIENTE-SERVIZIO: il primo riguardante il caso in cui ogni cliente usufruisce di un solo servizio, il secondo riguardante il caso in cui ogni cliente può usufruire di più servizi.

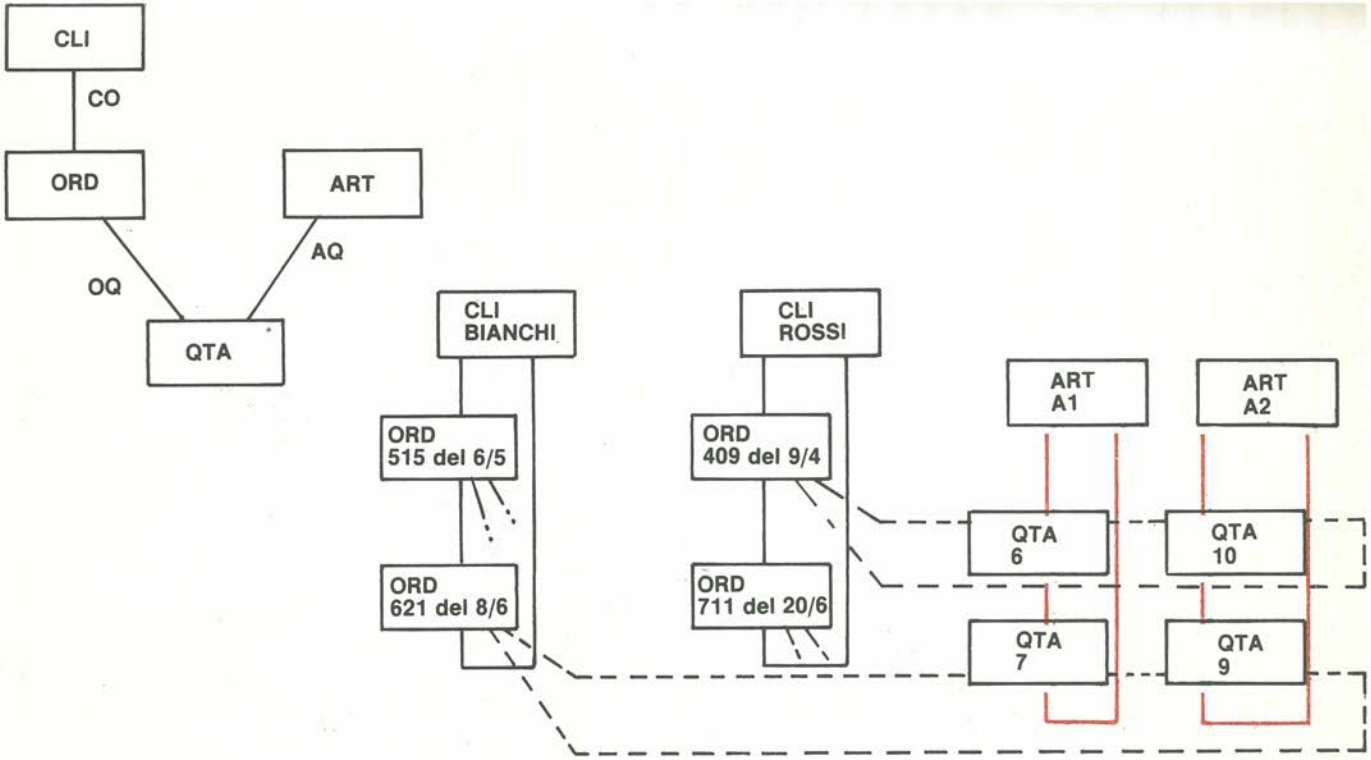


Figura 2.5.1 — Il problema delle duplicazioni nella struttura gerarchica è spesso risolto dal record connettore: ad esempio si osservi che nell'ordine n° 409 del 9 Aprile, il cliente Rossi ha ordinato 6 articoli tipo A_1 e 10 tipo A_2, gli stessi articoli sono stati ordinati da Bianchi nel 621, ma ciò non ha prodotto alcuna duplicazione di quegli ART.

Potrebbe sembrare che lo svantaggio della struttura con connettore consista nell'aggiunta di un record (non tanto della voce quantità, quanto proprio del record vero e proprio, con i suoi pointer) e dunque, almeno per numeri abbastanza alti, in un notevole spreco di spazio, ma non è così. In realtà normalmente lo spreco di spazio rispetto ad una struttura gerarchica risulta minimo se non inesistente. Inoltre, ci spiace ripeterci, ma è doveroso, la richiesta di procedura può comunque farci preferire questa soluzione proprio perché permette un accesso più facile su quel certo articolo (non più duplicato), o su quel certo ordine.

Comunque sia, se degli svantaggi ci sono, stanno da un'altra parte: si pensi ad una possibile distribuzione fisica dei record nelle pagine di Data Base e si osservi la figura 2.5.g.

Se abbiamo impostato le cose in modo da avere un record ORD con tutti i suoi QTA in una pagina, certamente, durante l'applicazione, il set OQ risulterà previlegiato, nel senso che sarà minimizzato il numero di accessi su questo set. Lo scorrimento del set AQ, invece, provocherà un alto numero di accessi. Ancora, certamente, le richieste di procedura avranno dovuto guidarci nella scelta di previlegiare CQ piuttosto che AQ, ma, in ogni caso, la scelta sarà stata alternativa.

Naturalmente il discorso è qui solo accennato, e non potrebbe essere altrimenti, ma si tenga presente che se la struttura a connettore presenta qualche svantaggio si tratta semmai proprio e solo di questo: in certe applicazioni il numero di accessi può risultare più elevato che con altre strutture come, ad esempio, la gerarchica.

Per concludere il discorso su questa struttura va ancora detto che la funzione di connessione non deve per forza essere rivolta alla sola individuazione di coppie: si osservi la figura 2.5.h, si potrebbero definire terne di record. Perché no, delle quaterne e via dicendo, dipenderà dai casi specifici.

Attenzione comunque al fatto che non sempre queste ultime strutture sono effettivamente comode o facilmente gestibili. Al proposito lasciamo al lettore la stesura di un longhand della prima struttura di figura 2.5.h che individui un certo numero di terne: ci si renderà conto che in molti casi essa non è della maggiore utilità, anzi a volte può essere preferibile stabilire più connettori diversi, o altri set.

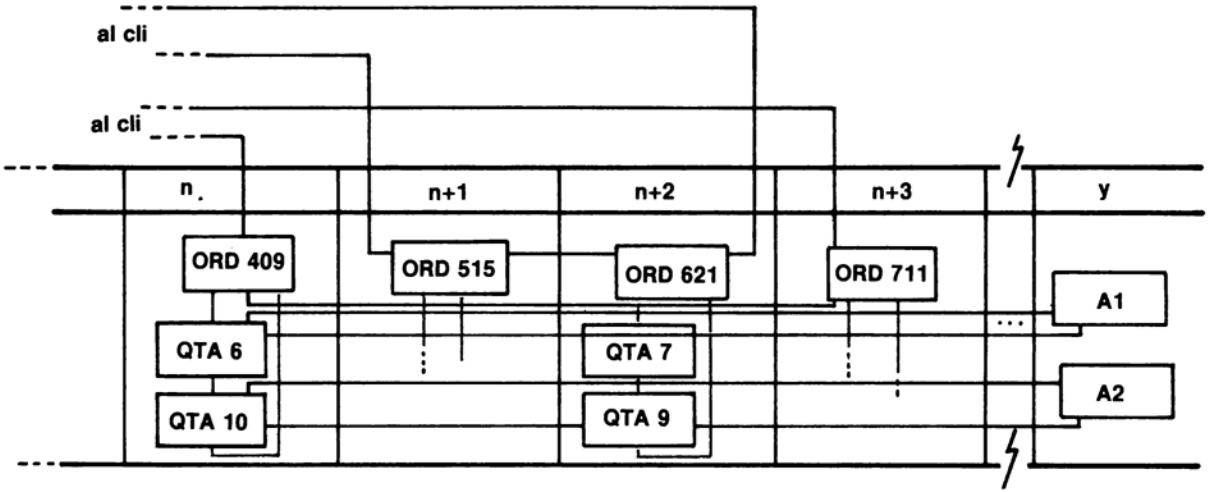
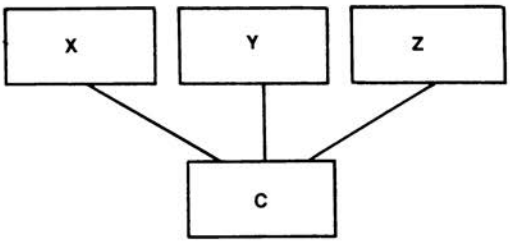
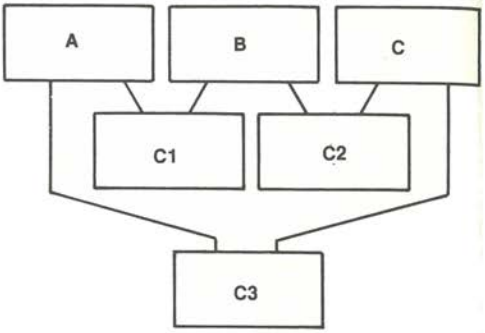


Figura 2.5.g — Una possibile disposizione fisica dei record del caso della figura 2.5.f: volutamente abbiamo tralasciato il record CLI per centrare l'attenzione sugli ordini ed in particolare sul 409 e sul 621. Il set OQ risulta privilegiato, ovvero il suo scorrimento provoca un solo accesso fisico poiché ogni catena singola è contenuta in una sola pagina. Lo scorrimento, invece, di AQ provoca vari cambiamenti di pagina, ovvero diversi accessi fisici: ad esempio partendo da A_1 in pagina y si deve andare in pagina n+2 quindi in pagina n per un totale di tre accessi fisici.





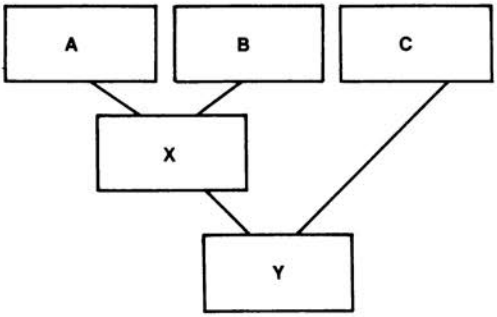


Figura 2.5.h — Alcune combinazioni delle strutture a record connettore: spesso si tratta di casi particolarissimi o di una certa complessità, certamente, tuttavia, il connettore permette non solo l'individuazione di una coppia, ma anche di terne, oppure di coppie con un altro elemento ecc.

2.6 Strutture reticolari: la reticolare doppia

La funzione di correlare due record esaminata nel paragrafo precedente ed affidata al record connettore, non si limita al caso di set diversi con owner diversi: essa può valere anche nel caso che l'owner sia dello stesso tipo. Si parlerà, allora di **struttura reticolare doppia**.

Si immagini di voler collegare tra loro due record dello stesso nome, o tipo, per via di una qualche relazione, ovvero di un legame logico di qualsiasi genere esso sia: se esistono due set diversi che collegano owner e member il problema è risolto in modo immediato.

Se, in altre parole, stabiliamo che *un owner sia tale in due set diversi con un unico member* realizziamo giustappunto, la funzione di correlare quei record tra loro pur essendo essi dello stesso tipo.

Si osservi la figura 2.6.a: i vari record A sono tra loro messi in relazione proprio per l'esistenza di un record B che stabilisce coppie di diversi, singoli, A.

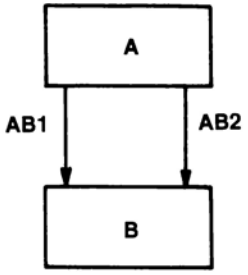


Figura 2.6.a — Struttura *reticolare doppia*. Il record di tipo B ha, ancora, la funzione di connettore: questa volta la coppia individuata da un certo B è dello stesso tipo record, si tratterà, cioè, di due record A.

Forse, per meglio comprendere il problema, varrà più, ancora una volta, un esempio. Al proposito ne esistono diversi ed abbastanza noti. Il primo degno di menzione ci sembra quello che chiameremo *dei voli*. Si immagini un'azienda di trasporti aerei (perché no, un'azienda di stato come l'Alitalia): essa vuole memorizzare la situazione generale dei voli, individuati da un codice, o numero di volo, in partenza da certe città ed in arrivo in un'altra. Il record CIT (che sta per *città*) sarà unico in quanto tipo e sarà collegato ad un altro (appunto del suo tipo) da un *connettore* che apparterrà a due set diversi: quello degli arrivi, ovvero *alla città* (ALLA-C) e quello delle partenze, ovvero *dalla città* (DALLA-C), come in figura 2.6.b. Dall'esame attento di questa figura già ci si può rendere conto del significato di *reticolare doppia*; la cosa va forse approfondita esaminando più da vicino il longhand. Prima tuttavia si

pensi alle minime differenze che esistono tra questa struttura, in quanto tale, e quella a connettore del paragrafo precedente. Si noterà che questa non ne è che un naturale sviluppo, se così vogliamo chiamarlo, e che il concetto di base, con tutto ciò che la questione della connessione implica dal punto di vista logico e fisico, rimane lo stesso.

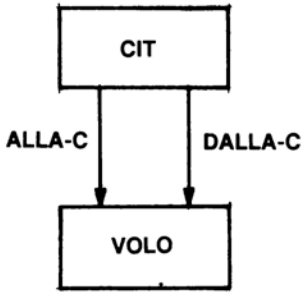


Figura 2.6.b — La *reticolare doppia* applicata al caso dei voli. Il codice, o numero di volo agisce da connettore di diverse città (CIT) e l'esistenza dei due set assicura il riconoscimento della località d'arrivo (ALLA-C) e di partenza (DALLA-C) di quel volo.

Ma si osservi il longhand di figura 2.6.c: se si vuol conoscere quale volo collega MILANO a ROMA è sufficiente scorrere il set DALLA-C partendo da MILANO per giungere, connettore per connettore, a quello che appartiene, nel set ALLA-C, all'owner ROMA ed il gioco è fatto. Ancora osservando attentamente questo longhand, si noterà come ognuno dei possibili collegamenti sia facilmente rappresentabile e come, dunque, il problema iniziale risulti risolto in modo facile ed immediato.

Quello della scelta del volo non è il solo problema risolto: ad esempio accedendo direttamente ad un dato volo è facile sapere da dove parte e dove arriva, oppure è possibile sapere quali sono tutti i voli in arrivo in una città e da dove provengono, e così via. Insomma i problemi risolti o risolvibili non sono pochi.

Come si è già detto, inoltre, questa struttura può servire alla soluzione di molti diversi problemi. Un altro assai tipico è quello delle squadre: si immagini di voler memorizzare il campionato di calcio. Basterà utilizzare due set, quello delle partite fuori casa (F-C) e quello delle partite in casa (I-C), con come unico owner la squadra e come member la data della partita. La situazione è riprodotta in figura 2.6.d e il longhand evidenzia una piccola parte di quello che potrebbe essere un ipotetico campionato di calcio.

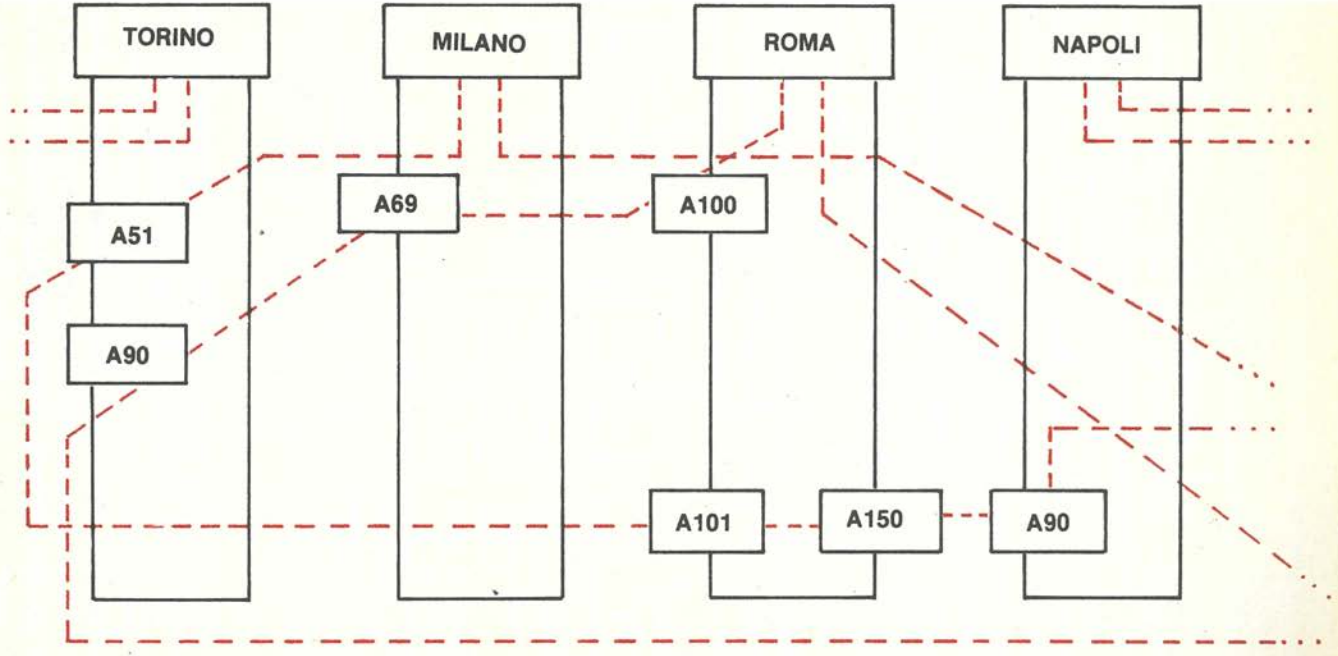


Figura 2.6.c — Un possibile longhand per la struttura di figura 2.6.b. Il set ALLA-C è disegnato in nero, DALL-C con linea tratteggiata rossa: volendo sapere quali voli partono da Milano per Roma basta, partendo dal record MILANO, seguire la linea rossa tratteggiata per trovare i codici di volo A51, da scartare poichè va a TORINO, poi A101 ed A150, da memorizzare poiché vanno a ROMA, poi ancora A90, da scartare, e così via.

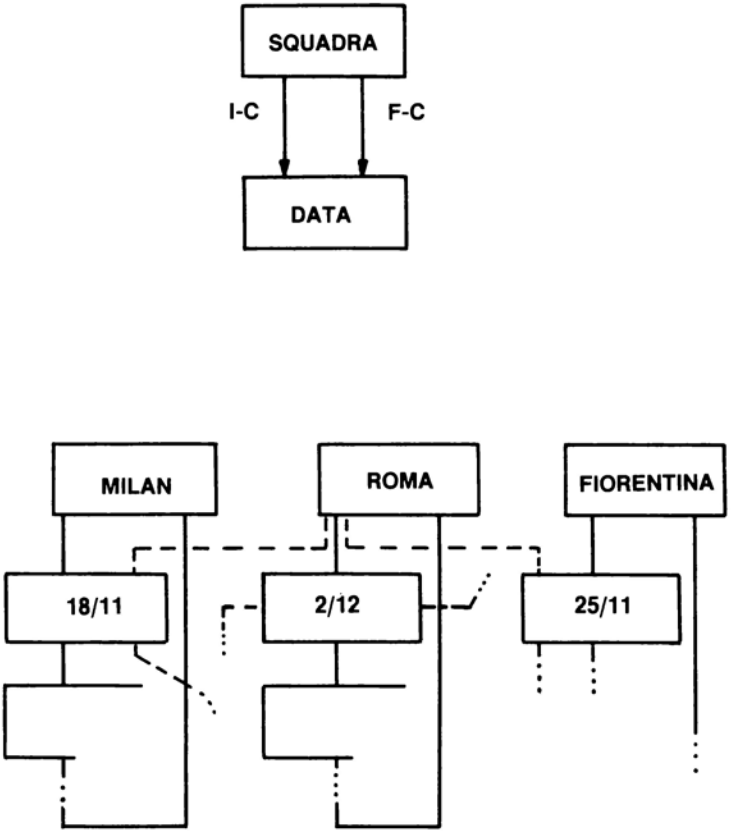


Figura 2.6.d — Alcune date di un ipotetico campionato di calcio: esse, inserite come contenuto del record connettore, stabiliscono il programma del campionato. Ad esempio, la data 18/11 stabilisce, in questa visione parziale, la partita tra il MILAN (owner nel set I-C, disegnato in linea continua ed indicante le partite in casa) e la ROMA (owner nel set F-C, disegnato in linea tratteggiata ed indicante la partita fuori casa).

Ancora è il disegno a spiegarsi da solo.

Un altro caso tipico, forse un poco più serioso, è quello della *distinta base*. Il problema è quello di memorizzare, in dettaglio, tutte le fasi di produzione di un certo articolo, evidenziando cioè, articolo finito, i vari componenti (e in quale quantità), fino ai componenti elementari.

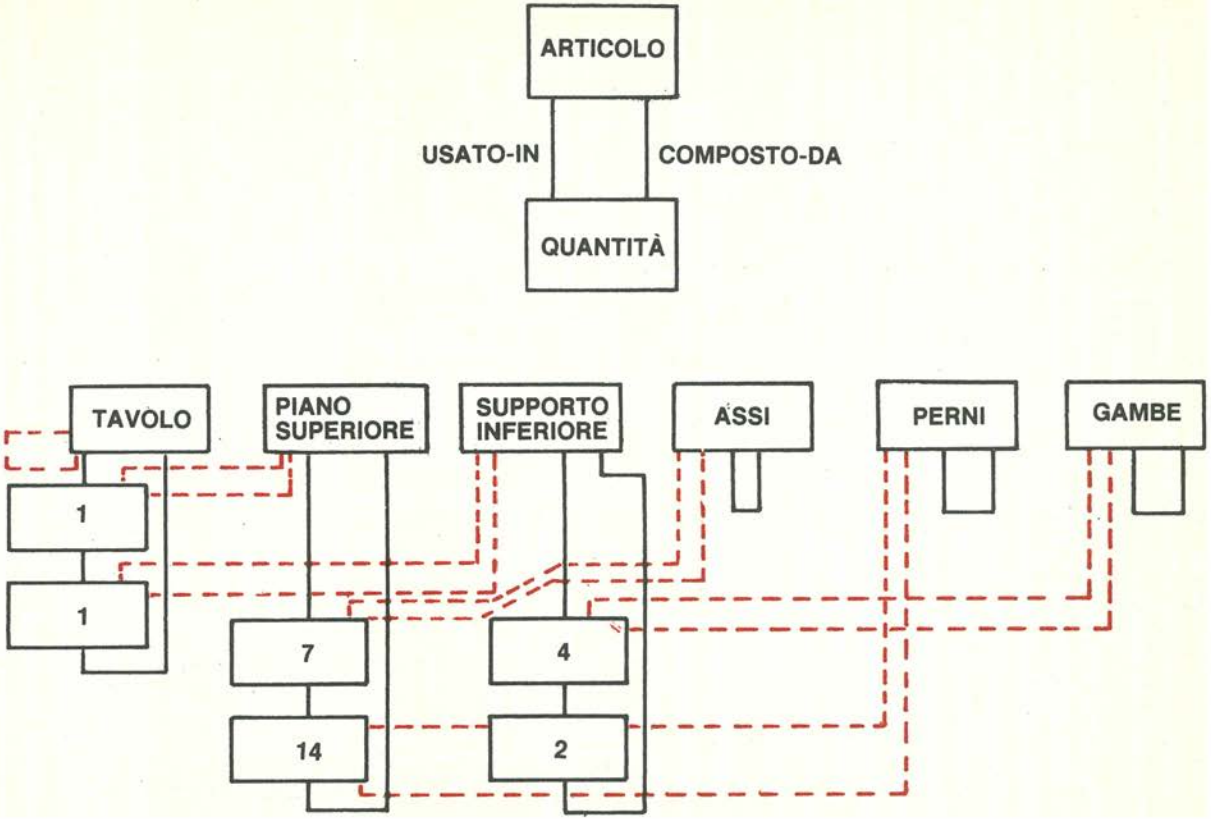
Immaginiamo una azienda che produce (scegliamo un articolo assai semplice) tavolini pieghevoli in legno, in assicciole come quelli da giardino. Tutto l'occorrente è: un certo numero di assi di legno ed un certo numero di perni (ad esempio quelli composti da dado e vite). Qual è tuttavia il miglior schema descrittivo della produzione? Ovviamente un qualcosa che tenga conto anche dei vari livelli intermedi (come potrebbero essere il piano del tavolo e la parte di supporto inferiore). Osserviamo la figura 2.6.e: abbiamo stabilito l'esistenza dei set USATO-IN e COMPOSTO-DA utilizzando il connettore per memorizzare quanto un certo elemento è utilizzato in un altro, intermedio o meno che esso sia. Si noterà che il prodotto finito ha il set USATO-IN vuoto, il set COMPOSTO-DA è vuoto invece per i componenti elementari (quali la vite e il dado).

Abbiamo volutamente esaminato un caso di una semplicità limite, ma pensiamo, ad esempio, ad un articolo come un frullatore: si provi, a mo' di esercizio, a riportare la seguente situazione sotto forma di longhand:

Frullatore	composto da	1 bicchiere			
	composto da	1 coperchio			
	composto da	1 rivestimento	composto da	8 viti	
			composto da	1 chassis	
			composto da	1 copr'motore	
			composto da	1 fondo	
			composto da	4 piedi	composti da 4 viti
					composti da 4 gommini
		1 motore	composto da	1 bobina	
				1 magnete	
				2 perni	
				10 viti	
				. . . .	

Le possibilità della reticolare doppia non si esauriscono con i tre casi visti: ancora a mo' di esercizio, osservando le strutture della figura 2.6.f si cerchi di disegnarne dei possibili longhand.

Ovviamente la prima è utile solo in casi particolarissimi, la seconda presenta una maggiore generalità (individua una realzione *coppia-elemento*); la terza ha la stessa funzione, ma con tutto ciò che implica l'aggiunta del connettore; la quarta, poi, permette l'individuazione di coppie di coppie.


ARTICOLO
USATO-IN
COMPOSTO-DA
QUANTITÀ
TAVOLO
PIANO SUPERIORE
SUPPORTO INFERIORE
ASSI
PERNI
GAMBE
1
1
7
14
4
2

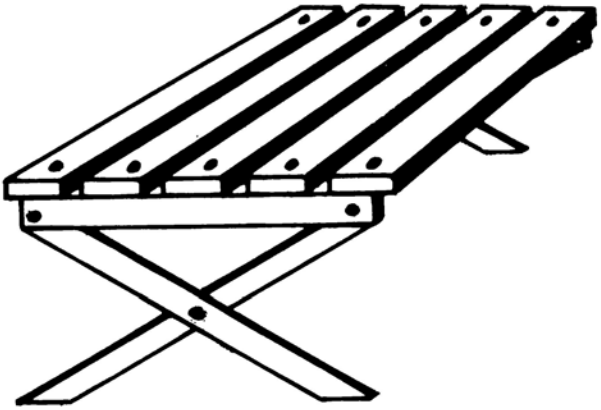


Figura 2.6.e — Distinta base, ovvero descrizione completa della composizione di un articolo: nel nostro caso un tavolo da giardino. Il set USATO-IN è stato disegnato in tratteggio rosso, COMPOSTO-DA in linea nera. Si noti come l'articolo finito (TAVOLO) abbia il set USATO-IN vuoto, mentre i componenti elementari (quali ASSI, PERNI e GAMBE) presentino vuoto il set COMPOSTO-DA. Si noti anche come certi componenti elementari possano essere usati in altri diversi componenti (i PERNI: 2 nel SUPPORTO INFERIORE e 14 nel PIANO SUPERIORE).

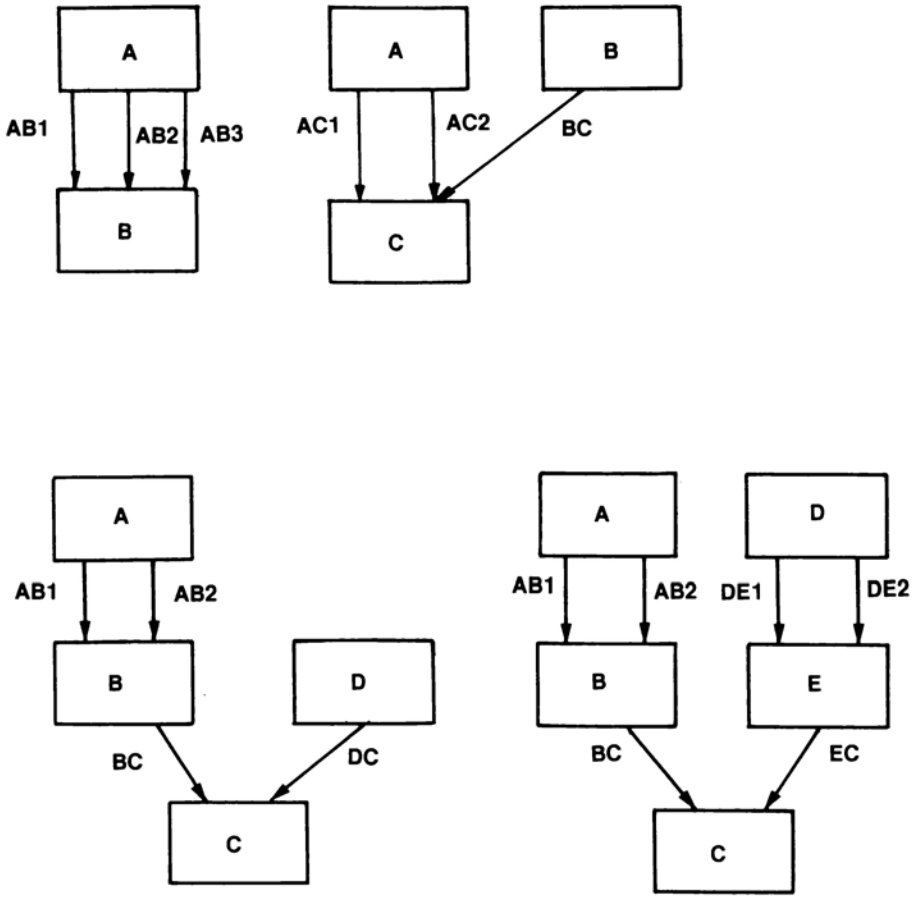


Figura 2.6.f — Le estensioni della reticolare doppia e l'uso del connettore comportano conseguenze di un certo rilievo quali l'individuazione di *terne*, di *coppie con elemento*, di *coppie con elemento* con *attributo* e di *coppie con coppie*.

2.7 Strutture reticolari: member-member

Esiste un ultimo tipo di struttura reticolare di particolare interesse: si tratta di quella che abbiamo voluto denominare **member-member** in quanto stabilisce *una relazione di dipendenza owner-member tra due record già member in altro set.* Ma vediamo più da vicino cosa significhi tale definizione. Nel paragrafo 2.3 abbiamo analizzato il caso del set con più tipi record. Ora, nulla impedisce che tra i member diversi dello stesso set venga istituita una nuova relazione gerarchica, ovvero che venga stabilito che un member è owner di un altro member in un nuovo set.

Osservando la figura 2.7.a si noterà come, pur essendo B e C member di A in ABC, il record B sia owner di C in BC.

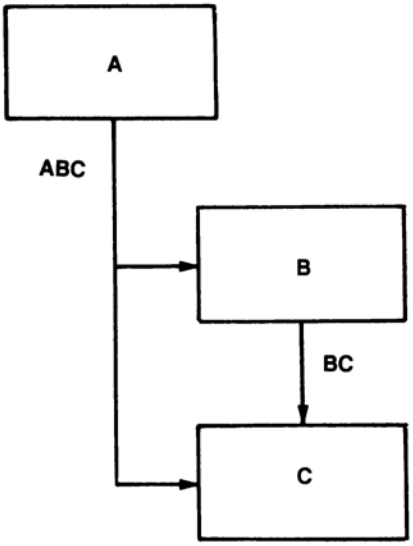


Figura 2.7.a — Il set ABC ha, come member, tipi record diversi: B e C. Tra essi è istituito un nuovo set in cui B compare come owner e C come member. Le relazioni tra A ed i suoi member (set ABC) e tra B e C (set BC) possono essere tra loro del tutto indipendenti.

La novità non è da poco per i risvolti pratici che possiede: semplicissima da un punto di vista strutturale è in grado di risolvere situazioni particolarissime, ma altrimenti ben difficilmente rappresentabili. Lo si comprenderà facilmente dagli esempi. Prima, tuttavia, è bene fare alcune altre precisazioni.

È facile immaginare come nel longhand del set ABC, si è cercato di chiarirlo sempre nel paragrafo 2.3, i record di tipo B e C siano tra loro mescolati in maniera indifferenziata. Nel caso che ora vogliamo esaminare esiste un fatto importante: i vari B sono, a loro volta, owner di alcuni altri C in un set diverso.

Non vogliamo, giunti a questo punto, analizzare un longhand già disegnato proprio perché è più importante comprendere la struttura come semplice base logica: si provi, comunque, e con pazienza, a farne alcuni disegni e si scoprirà come tra i record B e C, la relazione uno-molti propria della gerarchica non possa che convivere col set multiplo (a più member) a meno del caso di C che non possono *sopportare* un B quale owner (ed è ovvio!). Quest'ultimo caso potrà venire analizzato più oltre, ma ciò che ci preme in questo momento è come la struttura si adatti benissimo a quei casi in cui *certi dipendenti* di tipo diverso possiedono, tra loro, e a loro volta, una dipendenza del tipo uno-molti, ovvero gerarchica. Questo, ed è ancora una volta importante, deve farci considerare il fatto che in un eventuale disegno sceglieremo questa struttura sulla base di due principi: prima di tutto i B e i C dovranno *dipendere* dagli A contemporaneamente, in secondo luogo i B saranno *pochi* ed i C *molti* con, tra loro, una relazione di *dipendenza*.

Un'altra breve nota: anche questa struttura va ritenuta una reticolare. Essa infatti implica che le relazioni siano molteplici e che i record (almeno i B e C, per il caso della figura) siano *nodi* di più relazioni.

Sembrerebbe trattarsi di una combinazione di struttura gerarchiche, o di una ad albero con una gerarchica (ed in effetti forse così è). Tuttavia, vista nel suo insieme, e nella sua particolare tipicità, non si può far altro che notare come C rappresenti un nodo di connessione tra A e B e come B abbia una doppia funzione di dipendenza e di dominanza. Tutto ciò non può portarci che a ritenerla una tipica struttura reticolare.

Come al solito, meglio di ogni discorso teorico, può servire, per meglio comprendere sia l'aspetto logico che quello fisico della struttura, un esempio concreto.

A questo scopo riprendiamo un problema già affrontato: quello dei clienti di un'assicurazione che hanno stipulato certe polizze. Se, come nella figura 2.3.f, i clienti sono member delle polizze, un cliente che ne abbia stipulato un certo numero non può che risultare duplicato. Se, d'altro canto stabiliamo di far comparire ogni cliente come owner delle sue polizze, non potranno che essere queste ultime a risultare duplicate per tutti quei clienti che ne hanno stipulato dello stesso tipo.

La situazione reale potrebbe sicuramente fornirci delle indicazioni quantitative in modo da permetterci di scegliere il numero di duplicazioni minore, tuttavia il secondo caso permette di più. Ad esempio permette di memorizzare per ogni cliente altre cose quali (perché no?) i pagamenti a lui eseguiti a fronte delle sue stesse polizze. Questa è proprio la situazione del primo disegno della figura 2.7.b.

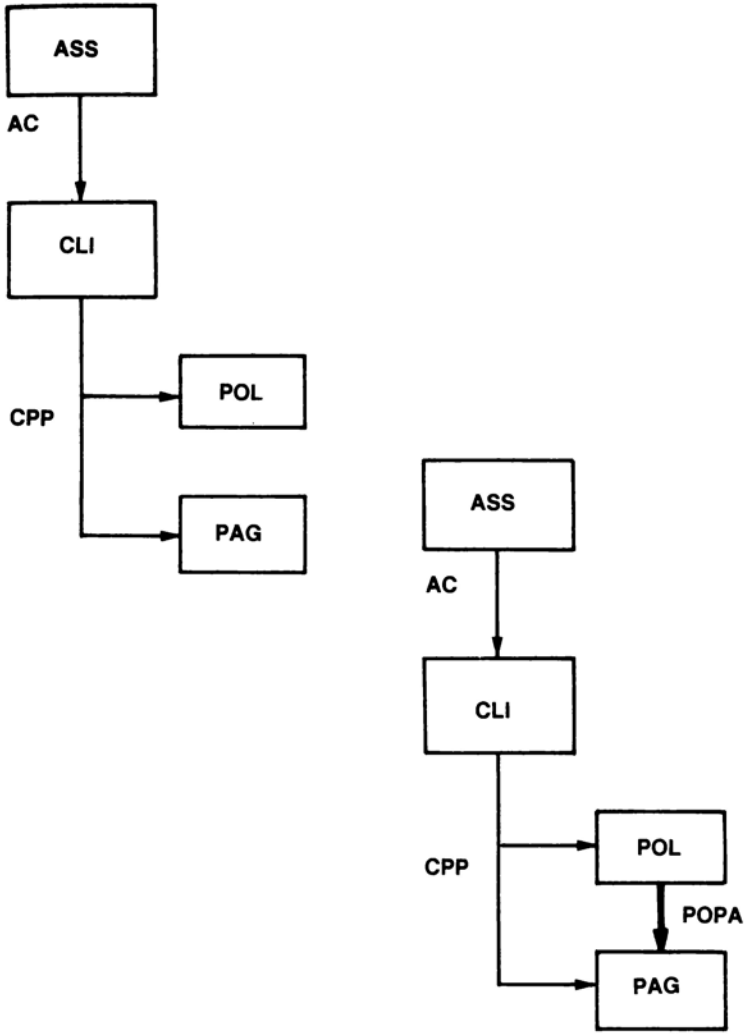


Figura 2.7.b — Il confronto tra i due disegni mette in evidenza le possibilità della struttura member - member: l'aggiunta del set POPA permette di memorizzare *per quali polizze sono stati fatti i vari pagamenti.*

La situazione è assai chiara: per ogni cliente è memorizzato il numero di polizze stipulato nonché di quali si tratti; inoltre ogni pagamento, appunto, ad egli fatto per causa di una sua polizza.

A questo punto si impone la domanda: *quale polizza?*. È nel rispondere a tale domanda che la struttura in esame dimostra le proprie possibilità: nel secondo disegno della figura appena vista (2.7.b), che corrisponde appunto alla nostra *member-member*, la questione è immediatamente risolta. L'istituzione del set POPA definisce con sicurezza a quali polizze siano dovuti certi pagamenti!

È chiaro che col disegnare POPA abbiamo rispettato i due principi cui prima si accennava: le polizze sono poche ed i pagamenti (*presumibilmente*) di più ed in secondo luogo, ma soprattutto, i pagamenti non possono che *dipendere* dalle polizze relative:

Ovviamente c'è di più: per memorizzare il pagamento fatto ad un certo cliente a fronte di una sua polizza, e qualora fossimo sicuri che è ciò che sempre accade, basterebbe il primo dei due disegni, ma non è sempre così. Potrebbe succedere che certi pagamenti vengano fatti a fronte di polizze di un altro cliente: ammettiamo, ad esempio, il caso di una polizza scudo, ovvero con pagamento per qualunque tipo di incidente, con colpa o meno, e soprattutto ammettiamo che si voglia rappresentare il caso di un pagamento fatto ad un cliente, ma causato da uno diverso (sia pure sempre dalla nostra assicurazione). Forse la figura 2.7.c mostra da sola come la *member-member* risolva il problema: prima di tutto si noterà come il cliente ROSSI abbia fruito di tre pagamenti di Lit. 100.000 prima, di Lit. 200.000 poi, a fronte della polizza casa, ed un terzo di Lit. 900.000 a fronte, questo è il secondo punto proprio il più importante, della polizza di un altro assicurato: il cliente BIANCHI.

Ci si obietterà che ciò vale giusto e solo nel caso di due clienti della stessa assicurazione e che, per di più, abbiano stipulato una polizza scudo: evidentemente ROSSI ha colpa dell'incidente in cui ha danneggiato l'auto di BIANCHI e la propria. Prima di tutto si noti che con lo stabilire POPA abbiamo memorizzato tutti i pagamenti che ROSSI, BIANCHI ecc. possono aver avuto e per quali polizze; in secondo luogo si noti che nell'esempio specifico si è stabilito per pura comodità che i due clienti appartenessero alla stessa assicurazione: nulla impedisce che nel set AC gli owner ASS siano di tipo diverso. È il caso di BOSSI assicurato per una polizza auto presso la società di assicurazioni A2 (che non è la nostra) e che ha provocato il pagamento di Lit. 650.000 a ROSSI.

Non è il caso di proseguire: sarà bene che, a questo punto, il lettore tenti da solo di intuire quali siano le conseguenze, sui grandi numeri, in questo caso specifico.

Le possibilità solutive vanno, ovviamente, estese ad una miriade di casi: il set potrebbe contenere più di due tipi record diversi, la cui dipendenza-dominanza permetta l'istituzione di altri numerosi set, potrebbe generare dipendenze multiple a loro volta definenti member-member di tipo particolarissimo. Si osservino in proposito i casi di figura 2.7.d, ma si tenga una volta di più presente che, le possibilità di combinazione sono vastissime e che una loro definizione non può che dipendere dalla comprensione della *logica* del problema che la pratica (o la richiesta applicativa) ci propone.

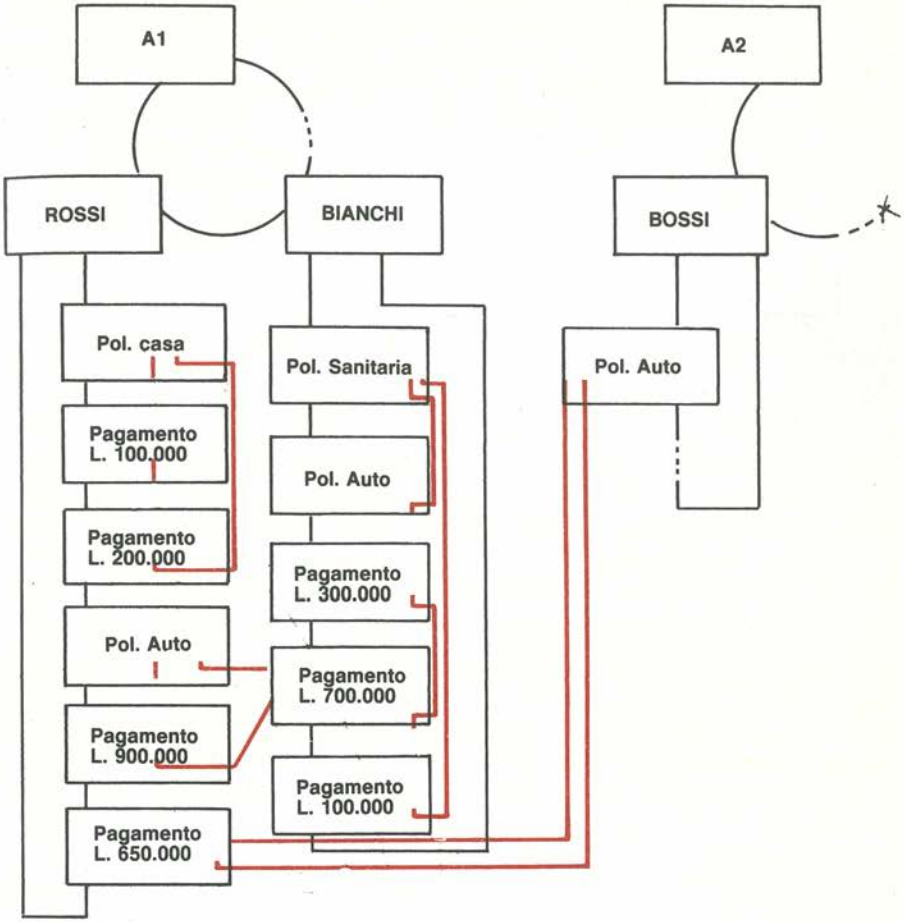


Figura 2.7.c — Un possibile longhand relativo alla figura precedente: sono presenti due Aziende di Assicurazioni (A1 ed A2), alcuni clienti, le loro polizze ed i pagamenti. Il set POPA (disegnato in rosso) evidenzia che, ad esempio, a ROSSI sono stati fatti due pagamenti a fronte della sua polizza Casa, ma anche un pagamento a fronte della polizza Auto di Bossi.

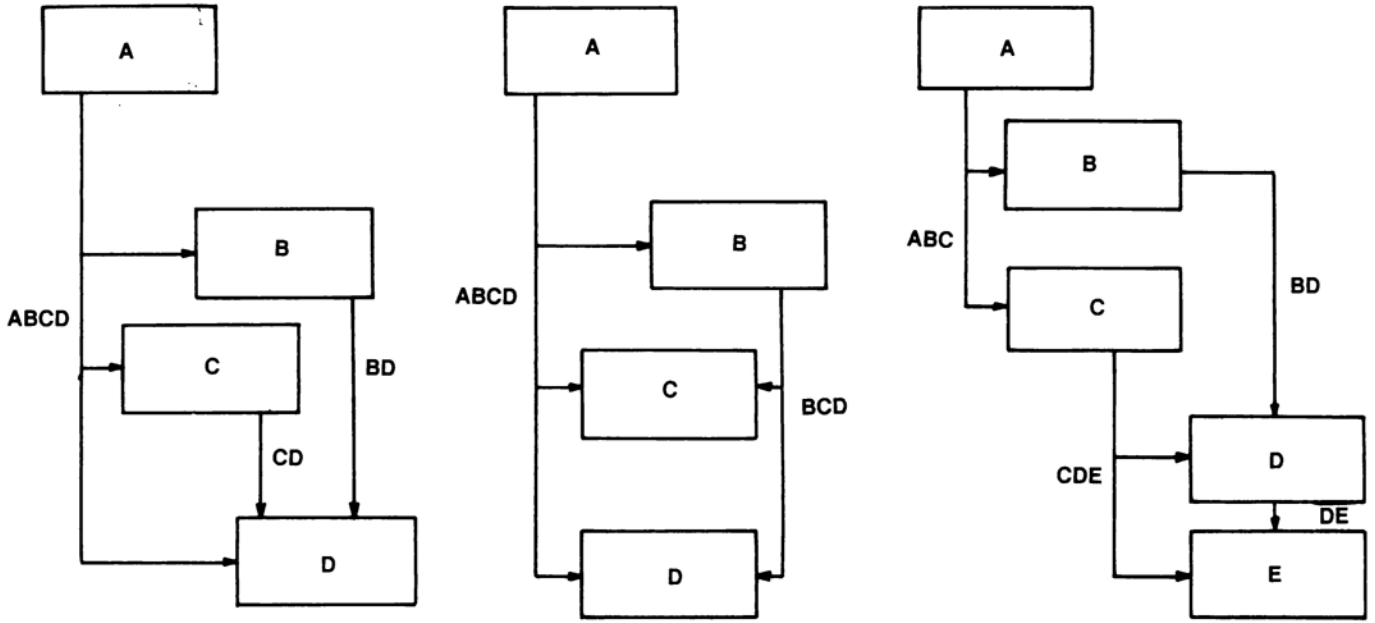


Figura 2.7d — Estensioni della struttura member-member: le possibilità sono ovviamente assai numerose ed il disegno può portare ad una *reticolare* di una certa complessità. Si noti anche come, spesso, queste estensioni comprendano, o facciano emergere ancora il concetto di connessione (D *connette* B e C nel primo e terzo disegno, ecc.).

2.8 Struttura ciclica e set opzionale

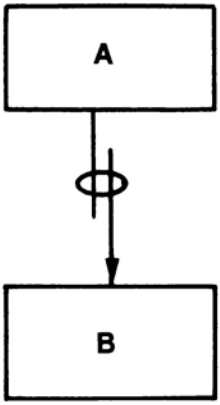
Nel Data Base Integrato oltre quelle viste esistono altre possibilità strutturali ancor più particolari, ma altrettanto importanti. Alcune di queste, nel nostro caso, vanno sicuramente citate, ma richiedono un esame preliminare di alcuni tipi di set specifici solo di alcuni sistemi, o, se si preferisce, di solo alcuni software di gestione.

Esaminiamo comunque tali casi poiché ne vale la pena: si immagini che un certo owner in un set possa *possedere* o meno dei member pur presenti, fisicamente, nel Data Base e chiamiamo tale set di tipo **opzionale**. Con ciò intendiamo che (si osservi la figura 2.8.a) la struttura primariamente disegnata come shorthand preveda dei record A come owner di certi member B; sottolineiamo la parola *preveda*: non è detto che, come è accaduto finora, un member B possa esistere solo qualora sia già presente il suo owner di tipo A. Osservando il longhand (sempre di figura 2.8.a) si noterà come B3 non appartenga a nessun A pur essendo esso presente.

Vanno tenuti presenti due concetti assai importanti in proposito. Primo: *i vari member (B) possono essere stati inseriti, o scritti fisicamente nel Data Base, senza che per questo fosse richiesta come necessaria la precedente presenza di un loro qualunque owner*; secondo: *la possibilità di collegare un member non connesso al suo owner (come B3) è comunque prevista proprio per come è fatta la struttura* (lo shorthand non fa preferenza tra, ad esempio, B2 e B3: non è possibile che il numero di pointer sia a priori diverso per questi due record e neppure, eventualmente lo si pensasse, per i record di tipo A), dunque potrà sempre risultare possibile, in un secondo momento, riconnettere ad un appropriato owner i record che sono stati inseriti senza collegamento, ma (perché no?) sarà anche possibile fare il viceversa, cioè sconnettere record precedentemente connessi.

Un'ultima notazione nei riguardi del set *opzionale*: sarà bene che i record che sono, o possono essere, sconnessi siano legati ad altro set, o comunque reperibili in qualche altro modo. Ci potremmo trovare, altrimenti, nella situazione di avere dei member perfettamente isolati (sconnessi) presenti fisicamente ed occupanti dello spazio senza che essi siano facilmente reperibili, quasi oggetti dimenticati e praticamente irraggiungibili!

Tenuto conto di tutto ciò vediamo alcuni dei casi in cui questo set *opzionale* può servirci: ne esistono di innumerevoli ed alcuni estremamente facili da comprendere. Si pensi all'esempio di figura 2.8.b: si vogliono registrare i settori di un'azienda i cui dipendenti hanno un'alta mobilità relativamente alle mansioni. Esponiamo meglio il problema: in un'azienda, si vogliono memorizzare tutte le mansioni e tutti i dipendenti, ma è importante poter cambiare frequentemente il dipendente di mansione, nonché poterlo lasciare presente senza che la sua mansione futura, terminata la precedente, sia ancora definita. È ovvio che il set opzionale ci aiuta proprio perché diviene estremamente semplice il cambiamento di mansione (e ciò è assai importante), ma anche perché risulta possibile l'esistenza del record dipendente senza che per questo sia definito un suo collegamento all'owner (la mansione).



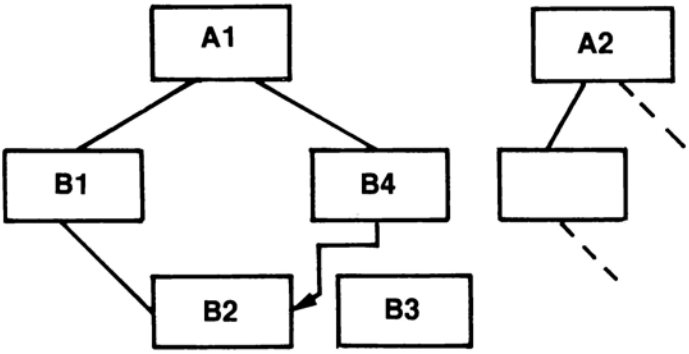


Figura 2.8.a — Shorthand e possibile longhand del set di tipo opzionale: i record member B possono essere collegati (come B1, B2 e B4) al loro owner A, oppure, pur fisicamente presenti, possono restare scollegati (come B3).

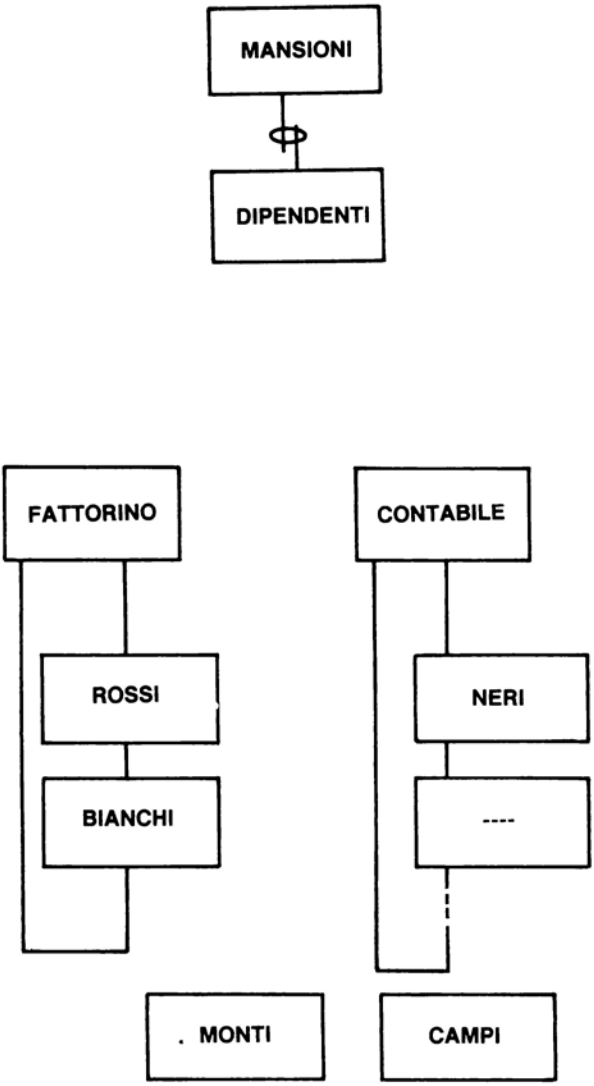


Figura 2.8.b — Il set opzionale permette di memorizzare situazioni particolari quali quelle dei signori Monti e Campi ai quali non è attualmente assegnata alcuna mansione.

D'altro canto se avessimo scelto di eseguire un disegno diverso in cui ogni dipendente dell'azienda avesse la mansione come record member, avremmo avuto due grosse incongruenze (o comunque svantaggi): prima di tutto il set tra dipendente e mansione avrebbe contenuto sempre un unico member (caduta la relazione uno-molti sarebbe caduta anche la ragion d'essere di una simile struttura gerarchica); in secondo luogo tutti i dipendenti con la medesima mansione vedrebbero una duplicazione dei loro member, o, meglio, più dipendenti aventi la stesssa mansione avrebbero set con member unico, ed uguale!

Non approfondiamo questi discorsi: si tenti però di disegnare, in quest'altro modo, un longhand corrispondente al caso citato e ci si renderà conto di quanto lo stesso disegno divenga più inefficiente proprio da un punto di vista logico, non diciamo poi da quello applicativo.

Tornando alla questione delle possibilità che questo set offre, va precisato che, tutto sommato, il caso visto non è neppure il più importante. Il set opzionale, infatti ci aiuta in alcune situazioni altrimenti non rappresentabili in alcun modo.

Affrontiamo nuovamente la memorizzazione di tutti i dipendenti vista nel paragrafo 2.3 (figura 2.3.c) pure tralasciando il record (unico) relativo all'azienda, ma pretendendo di registrare quali dipendenti dell'azienda sono *capufficio*... Prima di procedere si provi a immaginare qualche soluzione: ci si renderà conto di come né con l'aiuto di set multipli, né con quello di vari connettori si possa giungere alla soluzione. Il set opzionale invece ci risolve il problema in un sol colpo: è sufficiente istituire un set in cui il dipendente sia a propria volta owner del suo ufficio, ed il gioco è fatto. È ovvio che ciò è possibile proprio con un set opzionale, altrimenti ad un ufficio dovrebbero corrispondere più dipendenti e viceversa ad ogni dipendente più uffici! Col set opzionale invece, si osservi la figura 2.8.c, basterà che il dipendente prescelto sia trattato quale owner nel set opzionale (gli altri saranno sconnessi) il cui member sarà, giust'appunto, l'ufficio di cui egli è responsabile.

È questa la struttura che abbiamo voluto denominare **ciclica**. Essa, allora, è composta di due set: *ad una normale gerarchica è aggiunta un'inversione di dominanza per mezzo di un set opzionale* che permette l'esistenza di una relazione *uno di molti* a *uno o più*.

Il longhand mostra bene come, ad esempio, il dipendente Rossi che fa parte dell'ufficio Amministrazione ne sia anche il responsabile ed anche (lo si noti attentamente) come il Rag. Neri sia, pur facendo parte, a rigore, dell'ufficio ordini, responsabile di due uffici (ordini e acquisti).

Spiace ripetersi, ma è doveroso: ancora sono possibili applicazioni di tipo assai diverso ed estensioni notevolissime che possiamo lasciare quasi tutte all'intuizione del lettore.

Una sola di tali estensioni vale la pena d'essere subito esaminata per la sua peculiarità. Si pensi al seguente strano caso: essendo proprietari di una scuderia di cavalli di razza, vogliamo memorizzare di ognuno la sua discendenza e la sua prole. Il problema della *coppia* è facilmente risolto da un connettore. Si osservi la figura 2.8.d: il connettore C è un record vuoto che con la sua esistenza stabilisce una rela-

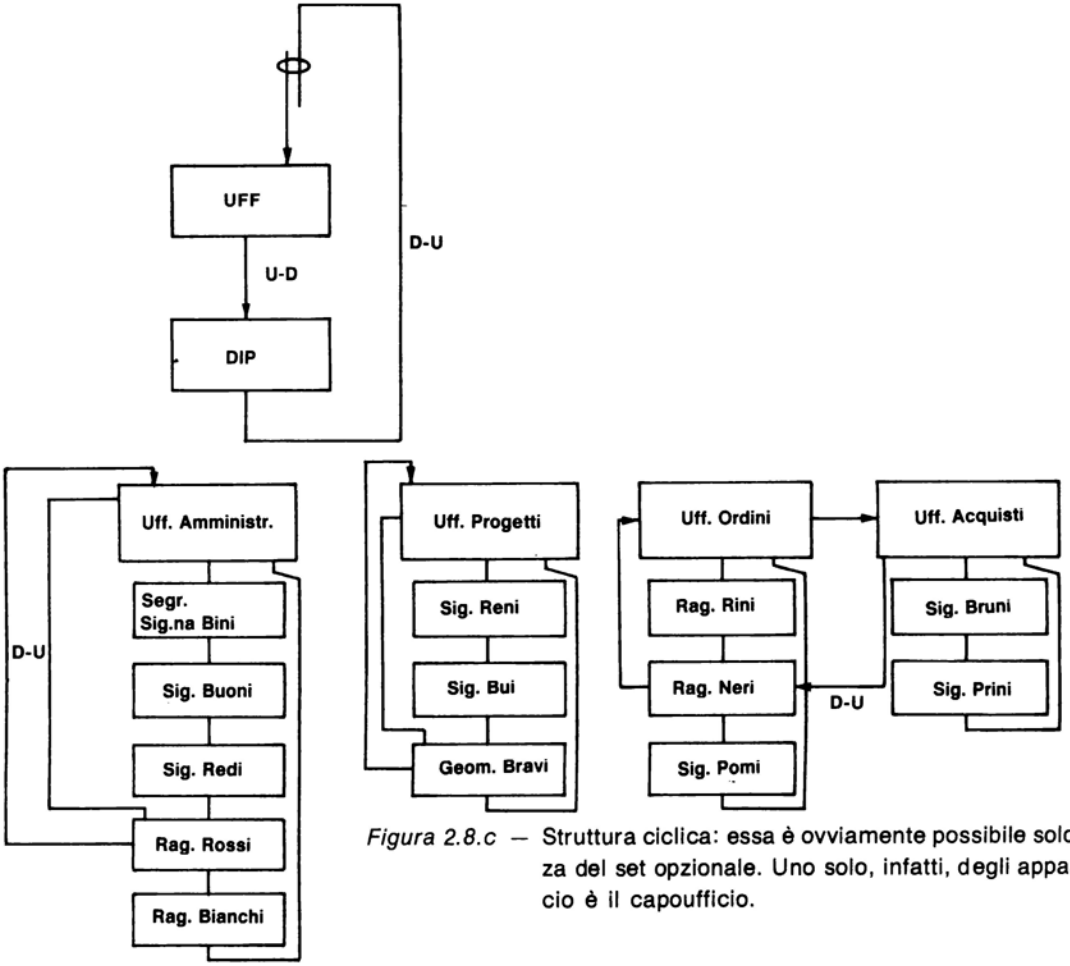


Figura 2.8.c — Struttura ciclica: essa è ovviamente possibile solo grazie all'esistenza del set opzionale. Uno solo, infatti, degli appartenenti ad un ufficio è il capoufficio.

zione tra due *genitori*, i record GEN, il set M è quello dei maschi, F quello delle femmine. Il *figlio* non è che un altro dei GEN: quello che appartiene nel set della prole PR al connettore che definisce la coppia generatrice.

Le possibilità non sono, ovviamente, evidenziate tutte nel longhand, ma si noterà come il problema risolto sia, in definitiva, quello della memorizzazione dell'*albero genealogico* e come, dunque, la sua applicabilità, a parte cavalli di razza o ronzini, possa risultare vastissima.

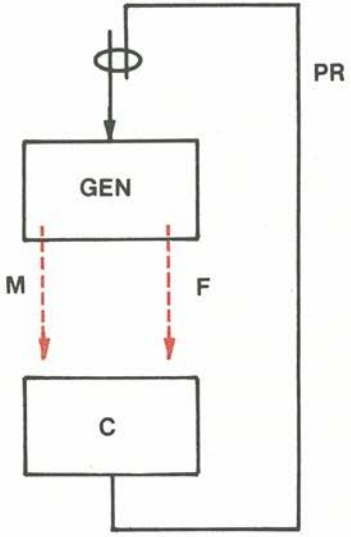
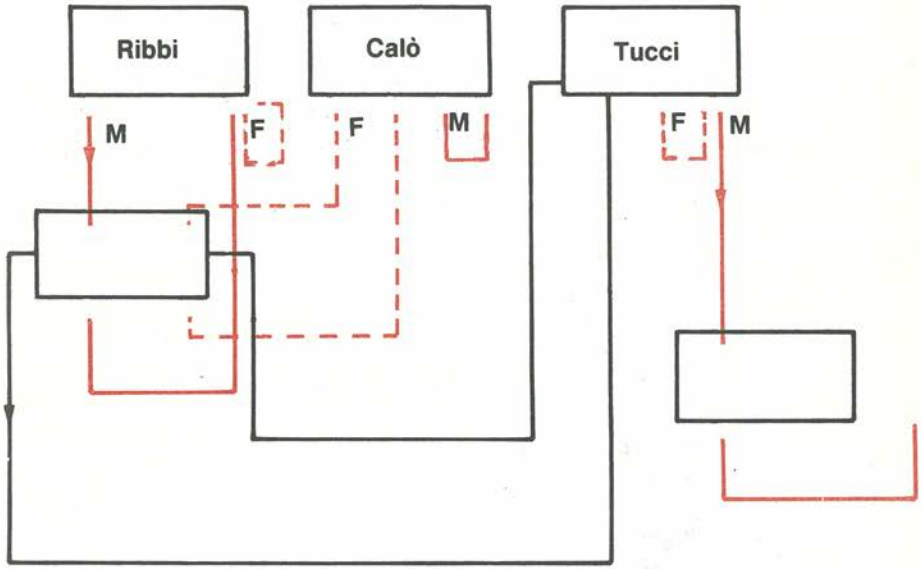


Figura 2.8.d — La struttura ciclica, realizzata sulla reticolare doppia permette la memorizzazione dell'*albero genealogico*.


Ribbi
Calò
Tucci
M
F
F
M
F
M

2.9 Estensione delle strutture base

Questo paragrafo non può che essere un sommario dei concetti già esposti, un'elencazione di casi più o meno particolari, ma direttamente conseguenti alle strutture di base fin ora viste. Ci si perdoni, una volta di più, l'insistenza: *le richieste di procedura sono determinanti al fine del disegno complessivo, ma esso non potrà che scaturire dalla combinazione più opportuna (appunto in funzione di quelle richieste) delle strutture di base.*

Vediamo quindi, prima di tutto, un riassunto schematico di tali strutture di base: si osservi la figura 2.9.a e si noti come in ognuna si sia scelto il caso più semplice e tipico. Si cerchi anche di ricordare il loro senso logico e di vederne le possibilità applicative.

Procederemo ora per esempi, anzi per disegni di strutture (magari confrontando le figure dei precedenti paragrafi) con l'intento di fornire un'idea un poco più precisa di cosa significhi quel concetto di *estensione* più volte ormai nominato.

Torniamo, allo scopo, al paragrafo 2.3 osservando, con attenzione, il problema di figura 2.3.f. Esso proponeva una struttura ideale qualora si fosse desiderato elencare tutti i clienti dell'assicurazione in funzione della polizza stipulata. Nel paragrafo 2.7 abbiamo visto, però, una modificazione sostanziale del problema: elencare le polizze in funzione dei clienti ed, in più, i vari pagamenti (figura 2.7.b). Estendiamo il problema: ammettiamo di desiderare, in sostanza, entrambe le cose, magari evitando quanto più possibile le duplicazioni.

Una struttura come quella di figura 2.3.f non può che duplicare i clienti che hanno stipulato più polizze. Una struttura, d'altro canto, come quella della figura 2.7.b, non può che duplicare le polizze. Magari essa non è di per se stessa causa di duplicazioni vere e proprie perché ogni polizza, ad esempio per il numero, è diversa da un'altra, tuttavia sarà pur sempre duplicato il tipo (il cliente Rossi ha la polizza auto, ma ce l'ha anche Bianchi come in figura 2.7.c). La soluzione, già lo sappiamo, la fornisce il record connettore. Forse si può tentare l'enunciazione di una regola di tipo generale: *quando i record di tipo dominante generano ridondanze sui dipendenti, si può introdurre un record connettore ponendoli entrambi dominanti, o owner, di quest'ultimo.*

Nel caso specifico dell'assicurazione perché non stabilire il connettore *numero-polizza*? I tipi di polizza non risulteranno più duplicati e la *indifferenza gerarchica* tra clienti e polizze ci assicurerà l'impossibilità di duplicazioni (v. figura 2.9.b).

Come si noterà le semplici strutture di base (gerarchica, member-member, e a connettore) si sono combinate generandone una più complessa che però risponde in modo assai soddisfacente al problema. Ci si obietterà che si sono introdotti set in più (e record in più), tuttavia bisogna tener presente che se il problema è ben risolto una qualche perdita, in termini di economia, non può che essere accettabile.

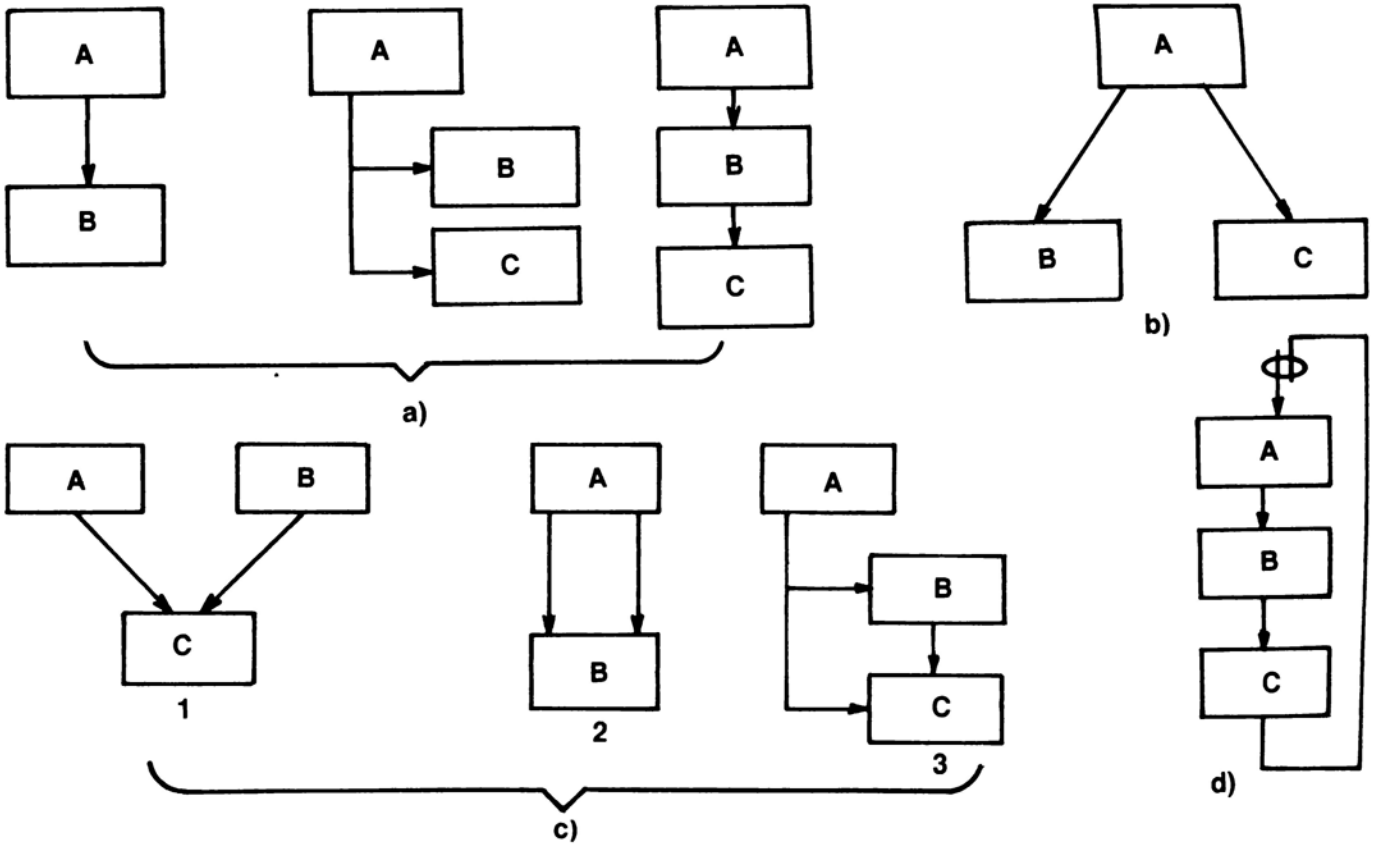


Figura 2.9.a — Panoramica delle strutture base: a) gerarchiche, b) ad albero, c1) reticolare a connettore, c2) reticolare doppia, c3) reticolare member-member, d) ciclica.

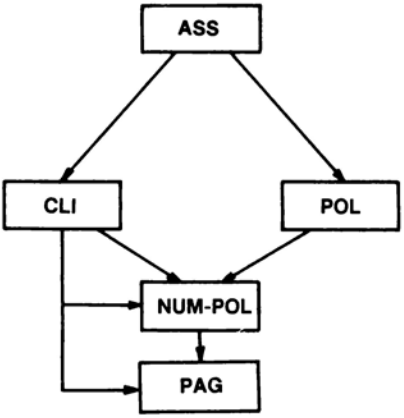


Figura 2.9.b — Il problema dei pagamenti e polizze di un'assicurazione può essere risolto combinando la struttura member-member con quella a connettore.

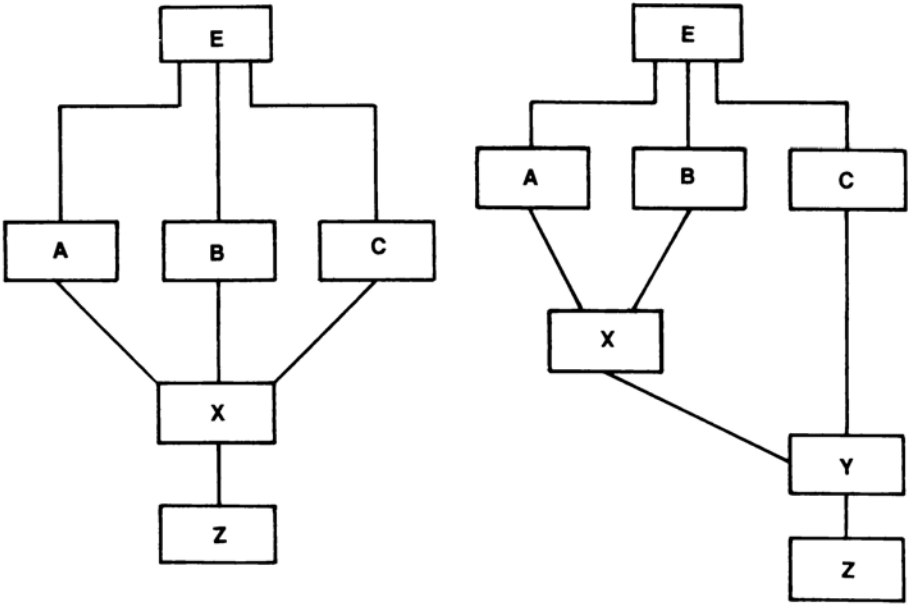


Figura 2.9.c — L'individuazione della *terna* può essere affidata ad un unico connettore (X del primo disegno): tale individuazione è possibile solo in casi particolari. L'individuazione, invece, della terna attraverso la coppia (con X del secondo disegno) e l'elemento singolo (con il connettore Y) è praticamente sempre possibile.

Quella vista è sicuramente un'estensione delle strutture base, o, se si preferisce, una loro combinazione. È ovviamente un'estensione in termini di *problema applicativo*, una combinazione in termini di *collegamento di relazioni di tipo diverso*.

Ma procediamo, magari con intenti più generali, ad esaminare qualche altro caso: in figura 2.9.c sono presenti due strutture che permettono l'individuazione di una terna (come in figura 2.5.h).

Quale delle due sia preferibile (o possibile) è ancora un problema di tipo applicativo: è ancora ciò che dobbiamo memorizzare e ciò che vogliamo ottenere che deve guidarci nella scelta. Ciò che è qui importante notare è che ancora si tratta di un'estensione della struttura a connettore, con in più (rispetto a figura 2.5.h) un record di *entrata* assoluta (E) ed una serie di informazioni (quelle del record Z) per ogni terna.

E che dire di figura 2.9.d? In questo caso il tentativo è quello di individuare delle quaterne.

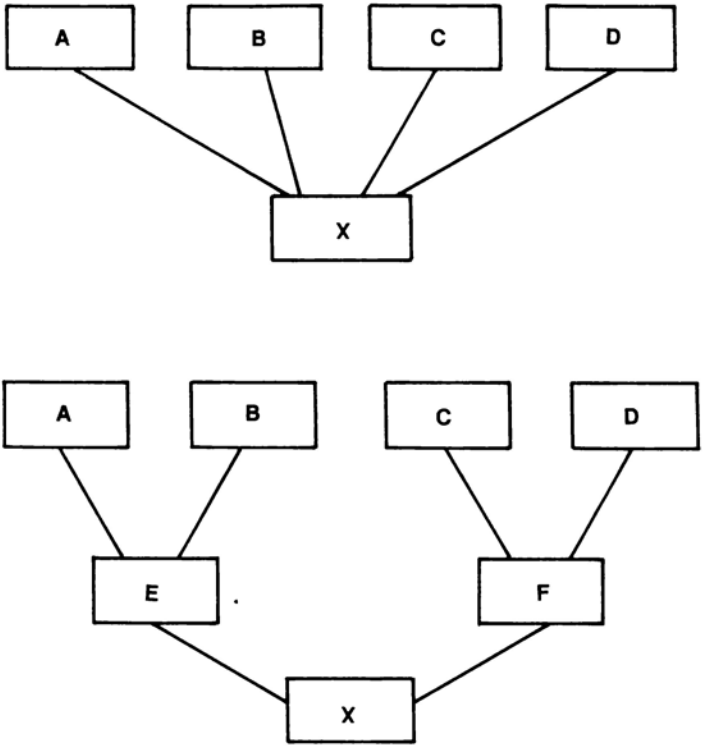


Figura 2.9.d — Estensioni della struttura a connettore: individuazione di una *quaterna*.

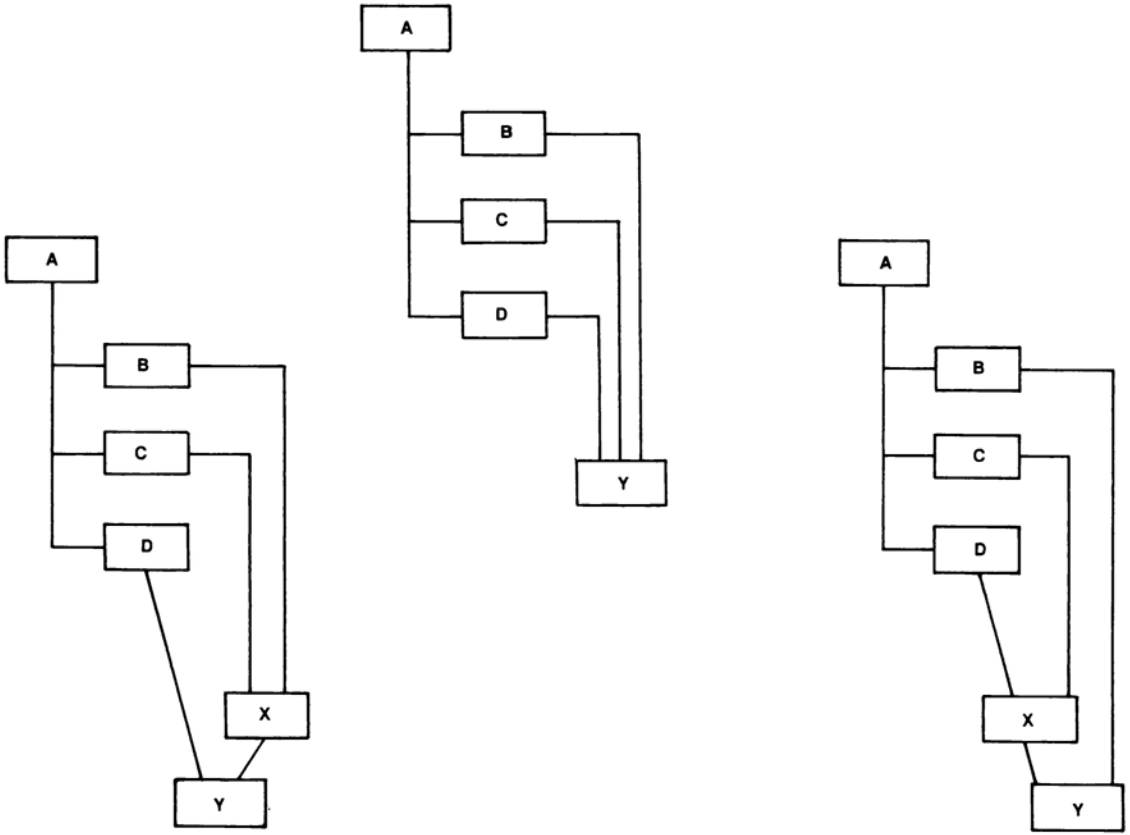


Figura 2.9.e — Estensioni della struttura gerarchica e a connettore: individuazione di *terne* dipendenti da un unico owner.

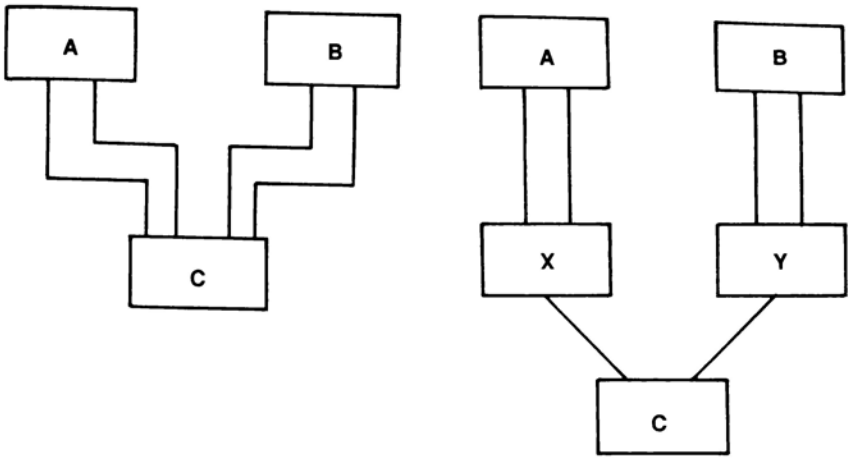


Figura 2.9.f — Estensioni della reticolare doppia e a connettore: si tratta ancora di una individuazione di quaterne.

La cosa, ovviamente, non è impensabile, ma attenzione al significato di questi disegni! Si tenti di individuare vari possibili longhand: ci si renderà conto di quanto la compilazione logica, indipendentemente dalla difficoltà di disegno, possa significare in termini applicativi (a volte l'individuazione della quaterna può risultare addirittura impossibile).

Un'altra estensione: può essere interessante connettere record di tipo diverso appartenenti allo stesso set. In figura 2.9.e l'individuazione di una terna è affidata a connettori combinati in vario modo.

Ancora: può rendersi necessario connettere delle coppie dello stesso tipo di informazione (non delle quaterne!). Si osservi la figura 2.9.f.

Lasciamo al lettore una interpretazione precisa di queste strutture, nonché, al solito, l'analisi dei possibili longhand: il campo d'applicabilità ai problemi reali non potrà che conseguirne.

Come si sarà ormai compreso si potrebbe procedere nel disegnare strutture ancora più complesse sempre combinando tra loro quelle di base e si potrebbe procedere, teoricamente, ad infinite estensioni. Un'analisi ben ponderata di ognuna di tali estensioni non ha senso che applicata a problemi applicativi concreti. Solo in tal caso risulterebbero evidenti i vantaggi e gli svantaggi di ognuna e la loro ragion d'essere.

Non è questa la sede per un simile lavoro, è tuttavia ormai facile intuire quale deve essere la strada da percorrere: affrontato un problema applicativo l'*invenzione* della struttura solutiva non può che dipendere dalla nostra capacità di combinare opportunamente le strutture di base estendendone e generalizzandone il significato.

2.10 Il disegno

Nel paragrafo precedente abbiamo già, implicitamente, descritto cosa significhi **disegno**. Forse è comunque giusto tentarne una definizione: *posto un problema applicativo, il disegno è la stesura in termini strutturali della soluzione.*

La definizione contiene in sè una miriade di implicazioni che è opportuno spiegare almeno per sommi capi:

a) il disegno è guidato dalle richieste di procedura;

b) la struttura complessiva è il *disegno*;

c) la stesura del disegno complessivo non può che dipendere dalla combinazione delle strutture base;

d) è l'inventiva (forse la fantasia) del *disegnatore* a permettere l'estensione delle strutture base a casi più o meno particolari.

Come si vede non si vogliono istituire delle regole assolute: non è questa la sede adatta a questo tipo di problema. Certamente, però, è possibile fornire dei consigli, od alcune regole relative, ma validamente accettabili nella stragrande maggioranza dei casi:

a) evitare, ove sia possibile, la ridondanza delle informazioni;

b) raggruppare i record in modo conveniente al fine di permettere accessi rapidi;

c) economizzare spazio evitando set (o pointer) inutili.

Ognuno dei tre punti elencati va ampiamente spiegato e, come al solito, cercheremo di farlo con degli esempi.

Per quanto riguarda il punto a) si ricordi quanto detto al paragrafo 2.5 a proposito del problema CLIENTE-ORDINE: si osservi ancora la figura 2.10.a: la prima struttura è sicuramente sconveniente rispetto alla seconda, con connettore, in termini di duplicazione, o di ridondanza, delle informazioni (l'articolo ordinato).

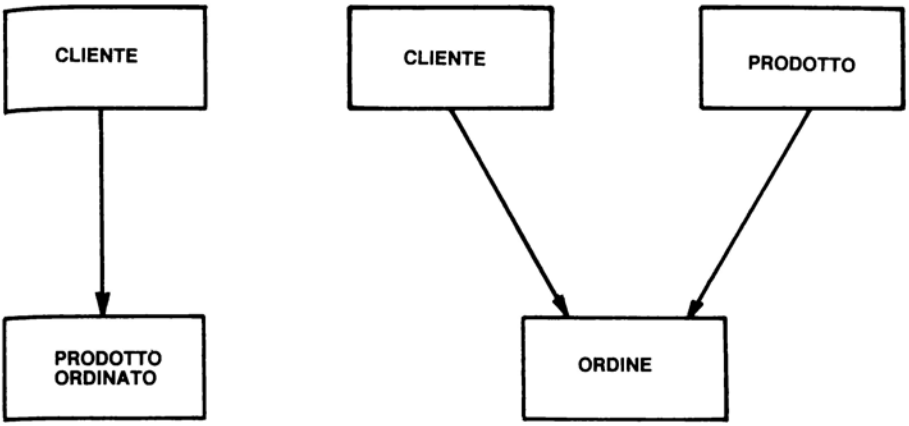


Figura 2.10.a — Nel primo caso il record relativo al prodotto ordinato viene duplicato per tutti quei clienti che hanno ordinato, appunto, lo stesso prodotto; nel secondo caso si ha un set in più, ma i clienti che hanno ordinato lo stesso prodotto sono allegati a quel record *unico* attraverso il connettore ORDINE.

Il discorso, al solito, non si esaurisce nel caso particolare: ogni volta che una struttura produce ridondanze, dobbiamo pensare a quali possibilità si abbiano di evitarle: quasi sempre il connettore ci soccorre utilmente. Se la ridondanza è dovuta alla gerarchica (è appunto quasi sempre così) possiamo essere sicuri che un confronto tra le due strutture ci condurrà ad un certo risparmio di spazio e ad una maggiore univocità di accessi. Sicuramente non è detto che siano queste le nostre maggiori preoccupazioni e può darsi che alla fine si decida comunque per la gerarchica, tuttavia il confronto non avrà potuto che aggiungere qualcosa al nostro primo tentativo di disegno, almeno in termini di conoscenza del problema applicativo che stiamo tentando di risolvere.

Per il punto b) poi, va detto che la maniera di raggruppare i record è spesso determinante ai fini di una soluzione efficace in termini di economia di accessi. Si pensi al problema di un'azienda che vuol memorizzare tutti i propri dipendenti nonché le varie valutazioni che vengono loro assegnate anno per anno: un disegno come quello di figura 2.10.b sembra l'ideale.

Esso, tuttavia, non può che, al solito, moltiplicare le valutazioni (quelle previste per numerose che siano, non potranno essere più di quattro o cinque, ma un dipendente, in funzione della propria anzianità, ne avrà assai di più): è il solito problema della pura e semplice gerarchica; ma che dire del confronto tra le due strutture di figura 2.10.c?

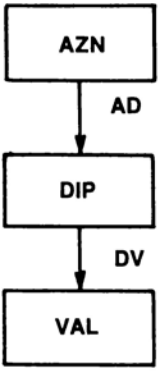


Figura 2.10.b — Il problema della gerarchica, che risolve semplicemente molte situazioni, è sempre quello delle ridondanze: le stesse valutazioni (VAL) sono ripetute per dipendenti (DIP) diversi.

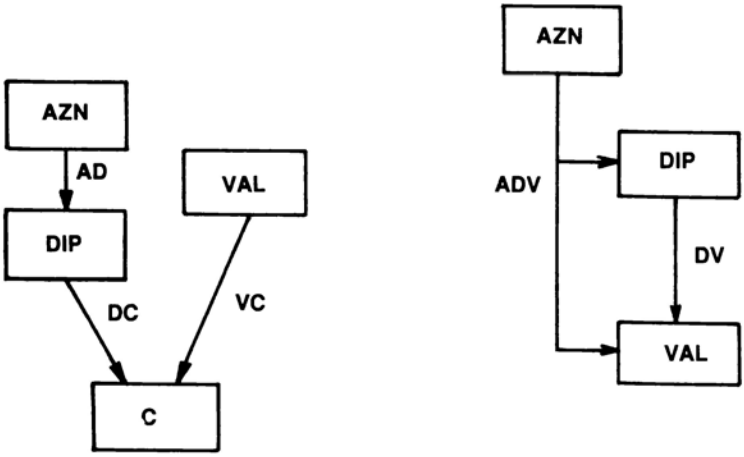


Figura 2.10.c — Il problema della figura precedente è facilmente risolto, al solito, dal connettore (C), ma questa volta può risultare preferibile la member-member ai fini di una maggiore facilità di accessi (ciò sicuramente implica l'accettazione della ridondanza delle valutazioni).

Nel primo dei disegni, sicuramente, le valutazioni non sono duplicate, però esiste la presenza di tre set: i record non possono essere raggruppati in maniera da occupare spazi vicini e, in applicazione, il numero di accessi non può che moltiplicarsi. Nel secondo disegno il *grappolo* dipendente dal record azienda è più facilmente *concentrabile*: abbiamo un set in meno e può essere studiato in modo di far stare tutto ciò che appartiene al set ADV in una sola pagina, minimizzando il numero di accessi fisici indipendentemente dall'esistenza del set DV.

Infine, per tornare all'elencazione delle regolette, il punto c): esso richiede che si evitino set e, di conseguenza, pointer inutili: è ovvio che, in tal caso, solo un conteggio aritmetico delle occupazioni può condurre ad una decisione definitiva. È altrettanto chiaro, tuttavia, che tra i due disegni di figura 2.10.d sia preferibile (se nulla osta in termini di richieste di procedura) il secondo: esso, infatti presenta un solo set nei confronti di tre.

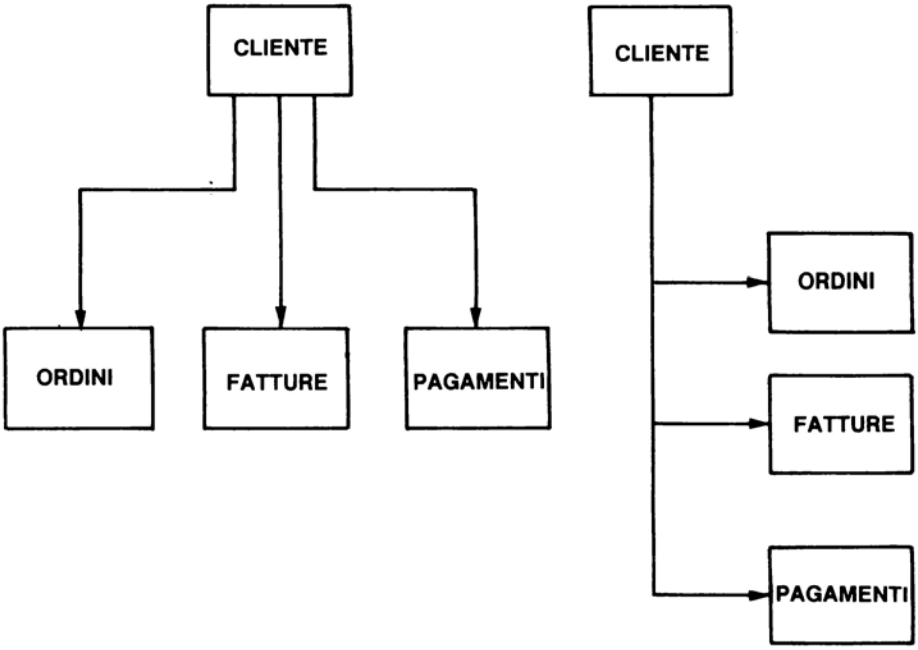


Figura 2.10.d — La scelta tra i due disegni (entrambi risolvono bene il problema della memorizzazione della situazione clienti) non può che dipendere da richieste di procedura (e dunque facilità di accesso) e valutazione di spazio occupato. Qualora si ritenga preponderante quest'ultimo aspetto ci si dovrà orientare al secondo disegno (meno set e dunque meno pointer).

Si potranno fare mille obiezioni a queste scelte, ma non si potrà negare che qualora il caso, od il problema, applicativo lo permetta ciò che è stato suggerito permette di risparmiare occupazione di spazio, o tempo di accesso in maniera sensibile: a noi la scelta di cosa vada previlegiato, ancora una volta il caso particolare non potrà che determinarla in modo definitivo.

Con tutto ciò, è ovvio, non abbiamo definito delle vere e proprie regole, semmai dei criteri di paragone; soprattutto abbiamo voluto suggerire quale sia la strada da seguire nel tentare la strutturazione di un disegno. Era l'idea, il concetto, di *disegno* che ci interessava e che abbiamo cercato di analizzare. La pratica, o l'esperienza non possono che essere le migliori maestre, tuttavia non si sbaglierà nel fare considerazioni simili a quelle esposte, anzi oseremmo dire che si sbaglierà a non farle.

CAPITOLO 3

DESCRIZIONI DEL DATA BASE E SUA REALIZZAZIONE

3.1 Il concetto di schema

Il *disegno*, abbiamo cercato di vedere di che si tratta, non è una cosa del tutto semplice da realizzare. Vanno tenute presenti molte cose e va fatto uno sforzo di inventiva non sempre facile. Tuttavia vi sono dei criteri applicabili nella generalità dei casi; forse, addirittura, esistono delle metodologie più o meno precise ed efficaci.

Un discorso introduttivo ai concetti di Data Base non è né adatto ad affrontare il problema, né permette la conoscenza degli strumenti appropriati ad intraprendere l'impresa. Limitiamo dunque il discorso a ciò che si deve fare dopo aver realizzato il disegno, partendo dal presupposto che le richieste di procedura ci abbiano comunque condotto ad esso.

La domanda che ci dobbiamo porre a questo punto è: come comunicare al sistema la nostra descrizione (o *disegno*)? È ovvio che dovrà trattarsi di una descrizione in linguaggio oggetto (ovvero *tradotta* o *compilata*); è altrettanto ovvio che dovremo fornirla allo stesso sistema in un linguaggio facile (magari *standard*): in un *sorgente descrittivo* che rispecchi tutte le caratteristiche e dettagli del nostro disegno.

Prima di procedere è necessario fare la conoscenza di un nuovo importante concetto adottato dal Codasyl: quello di **Schema**. Immaginiamo un disegno del nostro completo Data Base e definiamone la *descrizione* come l'insieme di informazioni che *comunicano al sistema la sua generale struttura, sia in termini logici che fisici.*

Descrizione logica significa:

- quali sono i record e di quali campi sono composti (nonché di che tipo: numerico, alfabetico, alfanumerico ecc.)

- come sono tra loro collegati (quale record è owner e quale member, quali sono i set e di che tipo, qual è l'ordine dei record nei set, ecc.).

Nonchè un'altra serie di informazioni strettamente dipendente dal sistema, quali il nome con cui intendiamo battezzare il nostro Data Base (o meglio la sua *descrizione*, o *schema*), la suddivisione delle strutture in differenti aree (o *file*) ecc.

Descrizione fisica significa:

- in quale quantità sono i possibili record per ognuno dei file che li dovrà contenere
- quale sarà la dimensione di tali file
- quale sarà la dimensione del blocco fisico ecc.

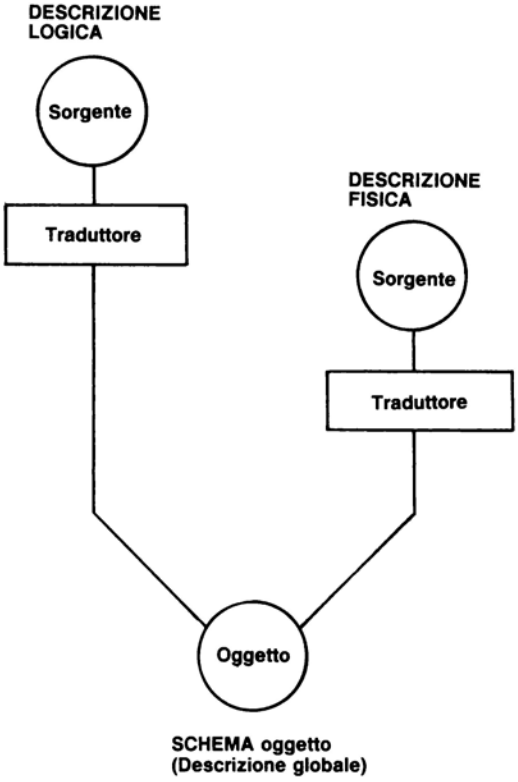


Figura 3.1.a — La descrizione logica viene scritta in linguaggio sorgente, tradotta dal sistema costituisce la prima parte di scrittura dello *Schema*. La traduzione della descrizione fisica, anch'essa scritta in linguaggio sorgente, completa lo *Schema* esso, in formato oggetto costituirà la descrizione definitiva, ovvero lo *Schema object* a cui tutte le attività (applicative o meno) sul Data Base faranno riferimento.

Le due descrizioni dovranno essere fornite separatamente al sistema, magari prima quella logica, poi quella fisica, per poi far parte della descrizione globale.

Esistono linguaggi appropriati per la scrittura in *sorgente* di tali descrizioni: nei prossimi paragrafi daremo alcuni limitati esempi degli standard relativi; per ora è importante comprendere bene il concetto che quella descrizione globale costituisce la base cui farà riferimento ogni altra attività di mantenimento del nostro Data Base.

Quella descrizione globale è lo **Schema**.

Si parla di *Schema logico* proprio intendendo la descrizione delle succitate caratteristiche logiche in linguaggio sorgente.

Si parla di *Schema fisico* intendendo la descrizione delle succitate caratteristiche fisiche in linguaggio sorgente.

Si parla di *Schema*, o *Schema object*, intendendo con ciò la descrizione globale già trattata dal sistema e, per così dire, già pronta all'uso.

Osserviamo, dunque la figura 3.1.a: ciò che dovremo fare, partendo dal disegno della struttura generale, sarà scrivere un sorgente logico che la descriva fin nei minimi dettagli, tradurlo (o compilarlo) producendo un primo output sullo Schema object; poi scrivere e tradurre (o compilare) la descrizione fisica, per ottenere quello finale.

Parrà, a questo punto, che questa fase preparatoria sia assai lunga. In teoria dovrebbe trattarsi di una cosa da farsi all'inizio e poi mai più; in pratica, accade che sia necessario rielaborare abbastanza spesso tali descrizioni. Sta di fatto, comunque, che queste necessarie attività descrittive sono rese assai facili dall'esistenza di linguaggi standardizzati semplici e molto efficaci.

3.2 Schema DDL

Il linguaggio che permette la descrizione in sorgente degli aspetti logici (e strutturali) prende nome di **DDL**, *Data Description Language*, ed ha formati e statement standardizzati dal Codasyl.

Vediamone, anzitutto, la struttura generale, ovvero la struttura dello Schema logico, o Schema DDL come lo chiameremo d'ora in avanti, senza troppo soffermarci sui dettagli, ma tenendo ben presente quanto detto in proposito nel precedente paragrafo. Teniamo anche presente che su sistemi differenti esisteranno particolari, altrettanto differenti specifiche: la parte comune sarà, tuttavia, preponderante. Uno schema DDL scritto per un sistema potrà venir facilmente compilato, previ pochi aggiustamenti, anche su un sistema diverso purché entrambi rispettino gli standard Codasyl.

La struttura generale, come si può osservare qui di seguito, è composta di vari gruppi (li abbiamo evidenziati lasciando più interlinee tra l'uno e l'altro) che chiameremo *entry*, le parti scritte in minuscolo sono quelle che dovranno essere sostituite dai nostri nomi, i trattini indicano i punti in cui possono essere presenti quei dettagli descrittivi che è impossibile approfondire in questa sede, i puntini di sospensione indicano la possibilità di ripetere la clausola o l'intera entry.

Diamo ora un'occhiata d'insieme al DDL; vedremo subito dopo il significato delle singole entry.

```
SCHEMA NAME IS nome-schema
  PRIVACY LOCK IS ------ ..... .
AREA NAME IS area1.
AREA NAME IS area2.
.....
RECORD NAME IS nome-rec1
  LOCATION MODE IS ------
  WITHIN areai
  PRIVACY LOCK ------
  -------.
20 campo1 TYPE IS ----.
20 campo2 TYPE IS ------.
RECORD NAME IS nome-rec2


...
SET NAME IS nome-set1
  OWNER IS nome-recj
  INSERTION IS ------
  ------.
  MEMBER IS nome-recj
  INSERTION IS ------ RETENTION IS -----
  DUPLICATES ------
  ------.
SET NAME IS nome-set2
  .....

  ...
END-SCHEMA
```

Come si può notare la prima entry, *SCHEMA-entry*, compare senza ripetizioni, è ovvio, e serve essenzialmente ad assegnare un nome logico, da noi scelto, allo schema. Nella stessa entry è anche possibile (è comunque facoltativo) definire delle protezioni attraverso l'enunciazione delle PRIVACY LOCK: cioè è possibile, per mezzo loro, definire delle *password* (o parole d'ordine) che dovranno essere dichiarate prima di accedere allo Schema-object. Senza tale dichiarazione esso rimarrà comunque protetto e illeggibile; non inserendo, invece, la clausola esso risulterà accessibile a chiunque ne conosca il nome.

La seconda entry, AREA-entry, fa riferimento ad un nuovo concetto: quello, appunto di *area*. Nello scorso paragrafo, ma anche precedentemente, si è già accennato al fatto che i vari record possono trovarsi su file fisicamente distinti. Tali record saranno comunque tra loro collegati (proprio per mezzo di pointer) ed il Data Base continuerà ad essere visto come un file logicamente unico. Ognuno dei file fisici viene chiamato *area*, o *realm* e va opportunamente battezzato con un nome ($area_1$, $area_2$,...).

La RECORD-entry, forse una delle più interessanti dal punto di vista delle possibilità di definizione, permette di descrivere le caratteristiche e la struttura di ogni record. Ne sarà presente uno per ognuno di essi. Prima di tutto si definirà il nome del record (nome-rec_i) e se ne daranno le caratteristiche: LOCATION definirà il modo in cui verrà fisicamente scritto, ovvero la sua posizione fisica. Ad esempio si potrà scegliere di porlo in una determinata pagina nella prima, seconda .... ecc., posizione; oppure si potrà scegliere di lasciare al sistema il compito di scriverlo nella posizione fisica più vicina possibile ad un suo owner, sempre che si tratti di un member; si potrà anche scegliere che la posizione debba essere determinata da un suo campo *chiave*, ecc. Come si vede le possibilità sono diverse e va detto che per lo più dipendono dal particolare DBMS anche se, tutte, hanno in comune quella definizione della posizione fisica che è determinante ai fini della migliore distribuzione dei record.

La clausola WITHIN permette di definire in quali aree i record si trovano.

La (o le ...) clausola di PRIVACY ancora permettono, come prima, la definizione di password di accesso, questa volta al record, o a certe funzioni (come lettura, scrittura, modifica) sul record stesso.

La seconda parte di questa entry permette la definizione dei campi del record: dichiarato un numero di livello (abbiamo utilizzato il 20, ma è permesso ogni numero maggiore di 1 e fino a 99), né più né meno come nel Cobol, si deve definirne il nome ed il tipo. La clausola TYPE permette la definizione dei campi alfanumerici, numerici, binari ecc... .

Un'altra entry di notevole interesse strutturale è la SET-entry: essa permette di definire, appunto, la struttura di ogni set e dovrà esserne presente una proprio per ogni set. Prima di tutto verrà definito il nome del set (nome-set_i), poi quale record ne costituisce l'owner, nonchè alcune sue caratteristiche. La clausola INSERTION, ad esempio, permette di definire una regola di inserimento dei record riguardante l'ordine logico che essi assumeranno nel set. L'ordine fisico, si badi bene, è già stato definito dalla LOCATION, ciò non toglie che il set, qualora lo si scorra seguendo le indicazioni dei pointer, possa condurre in posizioni diverse. Ciò significa che, ad esempio, si possa definire che un certo record che deve venir inserito, debba assumere la prima posizione dopo l'owner (ovvero che il pointer next dell'owner punti ad esso) anche se, in realtà, esso si trova, fisicamente, in una pagina, perché no, distante da quell'owner.

Poi andrà definito qual è il member del set (nome-rec_j), nonché le sue possibilità di connessione: INSERTIon e RETENTION (si ricordi quanto detto nel paragrafo 2.8) permettono proprio questo.

Esistono altre clausole: per fare un esempio, la clausola DUPLICATES permette di imporre l'impossibilità, o, viceversa, la possibilità, di certe duplicazioni dei record presenti in funzione dei valori di certi loro campi.

Siamo in un'area assai specifica e strettamente dipendente dal DBMS in questione e non vale sicuramente la pena approfondire la questione. Ciò che conta ora è la comprensione generale del DDL e della descrizione logica: potremmo concludere riassumendone in questi punti le caratteristiche:

- definizione del nome dello Schema;
- definizione delle aree che compongono il Data Base;
- definizione delle caratteristiche di ogni record e sua struttura in termini di sottodefinizione e tipologia dei campi;
- definizione strutturale di ogni set, ovvero definizione dell'owner e del member, nonché del tipo di set.

Sarà questa serie di informazione a costituire il sorgente della nostra descrizione e dunque ciò che dovremo tradurre e comunicare al DBMS: ciò costituirà la base di ogni passo successivo.

3.3 Schema DMCL

Gli aspetti fisici sono descritti in un linguaggio sorgente che prende nome DMCL, *Device Media Control Language*, e d'ora in avanti chiameremo **Schema DMCL**, appunto, la descrizione di come debba essere utilizzato il supporto fisico per l'immagazzinamento delle informazioni. In effetti il linguaggio di controllo sul Device Media (o *dispositivo di supporto*) serve soprattutto a definire quantità e dimensioni: *quantità* in termini di numero di record, *dimensioni* in termini di occupazione di spazio fisico su disco.

Anch'esso ha formati standardizzati dal Codasyl, tuttavia, come si può ben capire, in questo caso l'influenza del sistema su cui si lavora risulta determinante ai fini della stesura delle clausole. Il supporto fisico, il disco, ha *formattazioni* (*) specifiche, ma, soprattutto, è trattato differentemente dal software, ovvero dal punto di vista

(*) Il brutto termine, derivante dall'inglese, sta per *strutturazione*, o *organizzazione* dello spazio fisico ed implica una specifica maniera di trattare da parte del sistema (ed anche da un punto di vista hardware) lo spazio stesso disponibile sul disco.

della gestione del file (informazioni di controllo in generale sul file, suddivisione in blocchi fisici, o pagine, e relative informazioni di testata della pagina stessa ecc.). Ricordando che, per quanto ci riguarda, file è sinonimo di *area* (o realm) non possiamo far a meno di precisare che quanto segue è solo un esempio di cosa significhi descrizione fisica e che esso non può, proprio per quanto detto, essere completo.

Osserviamo, dunque, la struttura generale di un possibile schema DMCL; anche questa volta senza troppo soffermarci sui dettagli. Dovremmo poi analizzarne ogni clausola.

Si tenga presente che, ancora, ogni entry è evidenziata secondo il criterio del precedente paragrafo e lo stesso dicasi per quanto riguarda le clausole, i tratteggi e via dicendo...

SCHEMA NAME IS nome-schema.
AREA IS $area_1$
——— .
ALLOCATE n_1 DB-KEY
PAGE SIZE IS n_2
PAGE INTERVAL IS k
——— .
AREA IS $area_2$
.....
.....
....
.....

Non abbiamo evidenziato alcune entry spesso possibili e relative ad una peculiare descrizione dei set e dei record: da un punto di vista concettuale esse sono meno importanti e preferiamo soffermarci solo su quanto accennato.

Come si noterà, ancora la descrizione inizia con una dichiarazione di nome: quella dello schema e *nome-schema* deve essere lo stesso di quello dello schema DDL (non dimentichiamo che quanto stiamo descrivendo è qualcosa che aggiungiamo ad una descrizione già iniziata e che alla fine costituirà quello schema-object completo di cui si è già dato cenno).

Il concetto di area è assai importante, poiché, come si è già notato, significa *file* o *spazio permanente* su disco che manterrà le nostre informazioni; forse, proprio per questo, è l'entry più interessante in questa descrizione fisica.

Ad ogni area assegneremo un nome logico ($area_i$") che utilizzeremo nelle applicazioni. Per ognuna di esse dovremo dichiarare il possibile, e massimo, numero di record (n_1) in termini di DB-KEY.

Si ricordi quanto detto nel paragrafo ????: la DB-KEY (o Data Base KEY) non è altro che un numero intero che viene assegnato ad ogni record e che esprime, per il sistema, la posizione nel file. È, dunque, un indirizzo logico che viene trasformato dal sistema in posizione fisica nell'area: il primo record della prima pagina avrà DB-KEY=1; il secondo della prima pagina DB-KEY=2 e così via. E se supponiamo possano esistere 4 record per pagina, il primo della seconda pagina avrà DB-KEY=5, il secondo DB-KEY=6, il terzo 7, e così via. Non converrà mai, lo diciamo subito, utilizzare un così basso numero (4) di record per pagina proprio perché il prevederne di più non toglie niente alla capacità fisica vera e propria del nostro file: se provvediamo 64 DB-KEY per pagina e poi, per ognuna di esse, mettiamo solo 32 record, poco male: abbiamo solo sprecato *numeri*, non spazio effettivo.

Il numero n_1 è relativo a tutta l'area e per ora va tenuto presente, come già detto, che esso non è altro che il massimo numero di record che prevediamo di poter inserire nell'area.

Chiaramente si tratta di una previsione spesso difficile, va però tenuto conto del fatto che... *una qualche* previsione va pur fatta e che quanto più con essa ci avvicineremo alla realtà tanto meglio sarà!

La dimensione di pagina che va scelta con la clausola PAGE SIZE IS n_2, è intimamente collegata a quanto detto, ma anche alla clausola che segue, relativa al numero di record per pagina. Forse va prima osservato che n_2 è il numero di caratteri che si intende inserire in una pagina (si dovrebbe dire di *bytes*, restando con ciò coerenti al Codasyl), ovvero va detto che n_2 non è che una valutazione dello spazio che intendiamo assegnare ad ogni blocco fisico. Quel che è importante è che, ancora una volta, si tratta di un problema di dimensionamento e che questa volta la previsione è assai importante per quanto seguirà nelle applicazioni (si ricordi quanto detto a proposito degli accessi fisici). È chiaro che quanti più record stanno in una pagina, tanto più è facile che lo scorrimento dei set ci permetta di ritrovarli immediatamente (nella stessa pagina) senza un aumento degli accessi fisici stessi. D'altro canto non è possibile decidere per delle pagine *grandissime*: esse richiederebbero al sistema uno spreco di memoria eccessivo. Di qui l'importanza di mediare tra numero di accessi e dimensioni: ancora non esiste una regola precisa e non possono essere che l'intuizione e l'esperienza a guidarci in ogni caso specifico.

La clausola PAGE-INTERVAL, poi, non è che il completamento di quanto detto: essa definisce quanti record (k) possono, al massimo, trovarsi in ogni pagina. La mediazione di cui sopra non può che consistere nella determinazione di questo numero.

3.4 Il concetto di Subschema

Ogni applicazione *vede a modo suo* il Data Base.

In realtà ogni programma, o comunque modulo applicativo, raramente deve tener conto di tutto quanto il disegno (o, se si preferisce, della sua descrizione); normalmente, anzi, è solo una parte che risulta coinvolta e che contiene le informazioni che a quella applicazione servono.

Prima di approfondire questo concetto va detto subito che, ancora una volta, la maniera in cui l'applicazione *vede* il Data Base è strettamente dipendente dal sistema su cui si lavora.

Praticamente tutti i sistemi comunque ammettono l'esistenza di questo filtro, più o meno perfezionato, che fa si che siano le sole informazioni che interessano quelle di cui l'applicazione deve occuparsi, senza tener conto di tutto quanto il resto del disegno.

Nei software di gestione più avanzati esiste il concetto di **Subschema**: esso è una (ulteriore) descrizione delle informazioni che *interessano* una certa applicazione, sia che si tratti di un programma, sia che si tratti di moduli applicativi qualunque e più genericamente detti. Il Subschema può essere relativo ad uno, o più programmi: più applicazioni possono necessitare dello stesso *insieme* di informazioni e, dunque, far riferimento ad un certo Subschema, altri ad un gruppo di informazioni di tipo diverso e ad un Subschema, di conseguenza, diverso dal precedente.

Il termine programma od applicazione non deve trarci in inganno: esso come già suggerito, può significare moduli applicativi, o modalità applicative diverse.

Prima di approfondire questo concetto scendiamo, un po' più in dettaglio: definiamo **Subschema** quella *visione parziale delle strutture del Data Base che intendiamo utilizzare in certe applicazioni, o programmi, attenendoci alle caratteristiche del linguaggio applicativo da usare.*

Come si vede le cose importanti da notare sono due:

- la visione parziale
- il linguaggio applicativo.

Iniziamo coll'esaminare la prima: una volta che il disegno completo del Data Base è stato realizzato, non è detto che tutte quante le informazioni in esso contenute ci siano necessarie in tutti quanti i programmi (è quanto si accennava in precedenza). Anzi, di solito, certi programmi dovranno trattarne solo alcune ed il dover accedere a tutta quanta la visione di Schema potrebbe solo costituire una complicazione, sia in termini di visibilità (e dunque semplicità e brevità di programmazione), sia in termini di protezione (o esclusione delle informazioni che vogliamo proteggere, o mantenere private). Ecco dunque che una visione parziale ci facilita in modo determinante.

Ne riparleremo tra breve esaminando un esempio.

Per quanto riguarda il secondo aspetto (il linguaggio applicativo) va detto che ogni descrizione di Subschema è fatta in maniera aderente al linguaggio applicativo stesso: esistono, ad esempio, un Subschema Cobol ed un Subschema Fortran.

È soprattutto una questione formale (di scrittura degli statement come si vedrà), ma non solo. Nel disegno di un Subschema è determinante l'obiettivo che si intende ottenere: in una applicazione gestionale il linguaggio adatto (ad es. il Cobol) ci aiuterà ad un trattamento opportuno (definizione dei record e dei campi secondo criteri essenzialmente descrittivi), in un'applicazione tecnico-scientifica, invece, sarà determinante l'attenzione alla tipologia del dato in termini numerici (come il Fortran). Per di più la stessa struttura del Subschema (è ancora un problema di disegno, o, se si vuole, di *supervisione* del disegno) dovrà rispettare quelle richieste di procedura, di facilità di accesso alle informazioni, di trattamento, che, a seconda del problema, ci sono imposte.

Caliamoci, tanto per cambiare, nella realtà di un esempio. Nella figura 3.4.a si è voluto supporre che il disegno completo del nostro Data Base fosse, in definitiva, una combinazione delle strutture gerarchiche e reticolari più semplici.

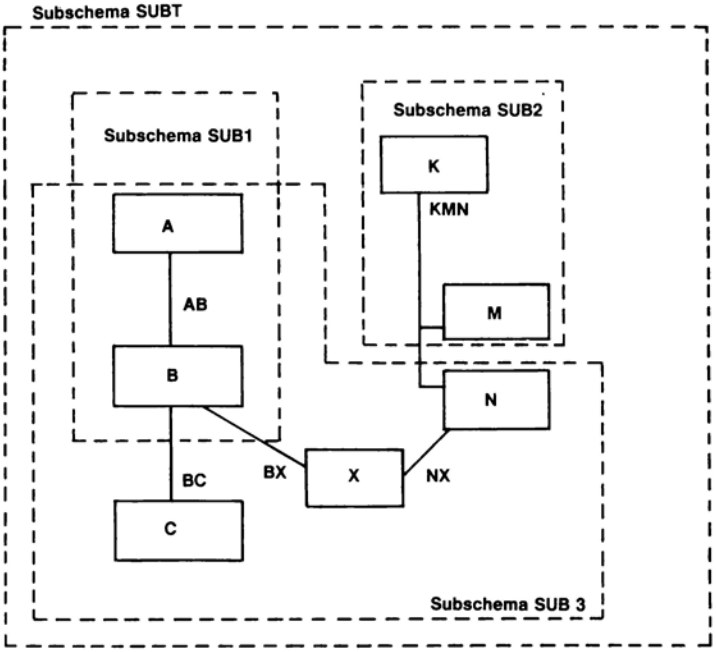


Figura 3.4.a — Il disegno completo è rispecchiato dal Subschema che abbiamo chiamato SUBT (Subschema totale), ma possono esistere vari tipi di altri Subschema, anche con parti comuni, capaci di unire gruppi di applicazioni diverse.

Ammettiamo ora che dall'analisi sia risultato che certe applicazioni, o per semplificare, un certo numero di programmi, necessitino di intervenire solo sui record A e B e sul relativo set.

Se tale gruppo di programmi *vede* solo tali elementi (come nel Subschema che abbiamo battezzato SUB1) si ottengono due ovvi vantaggi:

- la visione del disegno è semplificata rispetto allo Schema totale ed il programma diviene più facile ed immediato;
- le informazioni non appartenenti a tale Subschema rimangono protette ed il programmatore non può, né per errore, né per volontà, accedervi.

Non scendiamo nel dettaglio di come ciò possa essere diversamente realizzato sui vari sistemi; notiamo invece come tra SUB1 e SUB2 non vi siano elementi comuni: evidentemente si tratta di due gruppi di applicazioni sostanzialmente diverse (ma all'interno di ogni gruppo aventi caratteri simili). SUB3, che esclude invece solo il record K ed il set KMN, ha elementi comuni ad altri Subschema (gli stessi SUB1 e SUB2): ebbene ancora saranno state le caratteristiche applicative a farci decidere in tal senso. Insomma, siamo assai facilitati nell'aggregare liberamente i vari elementi proprio in funzione di quelle caratteristiche applicative e di quelle soltanto.

3.5 Subschema Cobol o Fortran e validazione

Si è notato, nel precedente paragrafo, come un concetto importante ed alla base di quello di Subschema, sia l'attinenza ed aderenza di quest'ultimo al linguaggio applicativo. Si è spesa qualche parola in più, forse, sulla questione della visione parziale, solo accennando al fatto che l'obiettivo applicativo è determinante ai fini della scelta del *disegno del Subschema*: non solo, esso è determinante anche ai fini della scelta del *tipo* di Subschema.

Di fatto un sorgente di Subschema va scritto in un linguaggio che, ancora, prende nome DDL (Data Description Language) e che è, appunto, strettamente aderente al linguaggio applicativo che si intende utilizzare.

Esistono almeno due tipi importanti, sui sistemi più specializzati, di Subschema: quello Cobol e quello Fortran. Se, per tornare alla figura 3.4.a, il Subschema SUB1 deve essere utilizzato in applicazioni di carattere gestionale, per le quali è sicuramente il Cobol a rivelarsi il linguaggio applicativo più adatto, la sua descrizione dovrà essere fatta con il **DDL Cobol**.

Nulla impedisce, tuttavia, che, se SUB2 è rivolto ad applicazioni tecniche o scientifiche, di esso venga fatta una descrizione in **DDL Fortran** ed il tutto a fronte dello stesso Schema.

Aggiungiamo subito che, come si vedrà dagli esempi di DDL che seguiranno tra breve, la differenza tra i due linguaggi è quasi puramente formale: si tratta di maniere diverse di scrivere più o meno le stesse cose. Lo stesso trattamento ed allocazione dei Subschema da parte dei programmi procederà secondo criteri identici. Ciò che, semmai, rappresenterà una differenza importante, consisterà nella definizione del *contenuto* dei record, nella tipologia del dato e nella sua formattazione.

Prima di procedere vediamo un esempio della struttura generale dei Subschema DDL dei due linguaggi.

Iniziamo con il più diffuso DDL Cobol. Come il Cobol, il Subschema è strutturato in DIVISION e SECTION ed è composto di vari paragrafi (la scrittura che segue usa le stesse notazioni dei paragrafi relativi a Schema DDL e DMCL):

TITLE DIVISION.
SS SUBSCHEMA IS *nomess*$_1$ WITHIN *nome-schema.*

MAPPING DIVISION.
ALIAS SECTION.
AD *nome*$_2$ BECOMES *nome*$_3$.
AD *nome*$_4$ BECOMES *nome*$_5$.
.....
STRUCTURE DIVISION.
REALM SECTION.
RD *nome-area*$_1$.
RD *nome-area*$_2$.

.....
SET SECTION.
SD *nome-set*$_1$.
SD *nome-set*$_2$.

.....
KEY SECTION.
KD *nome-K*$_1$.
KD *nome-K*$_2$.

.....
RECORD SECTION.
01 *nome-rec*$_1$.
    02 *nome-campo*$_{10}$ PIC ..... .
    02 *nome-campo*$_{11}$ PIC ..... .

01 *nome-rec*$_2$.
02 *nome-campo*$_{20}$ PIC ..... .
02 *nome-campo*$_{21}$ PIC ..... .

.....
.....
END

La divisione TITLE DIVISION serve praticamente solo a definire il nome Subschema, a battezzarlo in qualche modo: *nomess*$_1$, ed a precisare a quale Schema esso fa riferimento: *nome-schema*.

La MAPPING DIVISION è presente per permettere eventuali sostituzioni di nomi: se, ad esempio, *nome*$_2$ è stato usato nello Schema per un qualche elemento del nostro Data Base, nelle applicazioni che riferiranno a questo Subschema, dovrà essere usato il nome *nome*$_3$; *nome*$_5$ al posto di *nome*$_4$, ecc.

La STRUCTURE DIVISION è quella su cui va appuntata la maggior attenzione: essa ci permette di definire quali elementi dello Schema entreranno a far parte del nostro Subschema.

In REALM SECTION elencheremo tutti i nomi delle aree (Realm è sinonimo di Area) che entreranno a far parte del Subschema, ognuno sotto il nome paragrafo RD.

In SET SECTION elencheremo i nomi di tutti i set che parteciperanno al Subschema.

In KEY SECTION elencheremo i nomi di tutte le chiavi di accesso permesse dal Subschema (in molti sistemi è possibile che certi campi dei record vengano definiti *campi-chiave* per permettere un accesso diretto in base al loro contenuto).

In RECORD SECTION elencheremo i record che faranno parte del Subschema. È in questa sezione che, forse più che nelle altre, si vede l'attinenza al linguaggio: ogni record è descritto compiutamente, con le sottodefinizioni dei campi con i loro numeri di livello (si è usato 02 solo per semplicità) e con la relativa PICTURE proprio come in COBOL. Un campo definito con una sua tipologia numerica o alfanumerica in Schema, diverrà qui di PICTURE X(...) o 9(....)....

Tutti i nomi qui definiti saranno utilizzabili nel (o nei ...) programma Cobol che a questo Subschema farà riferimento; solo quelli saranno richiamabili e lo Schema diverrà del tutto trasparente: il programmatore dovrà basarsi solo su questa descrizione.

Il DDL Fortran ha caratteristiche del tutto simili. Come si può vedere da quanto segue il significato è identico, ma le varie elencazioni dei nomi e delle strutture dati sono scritte in maniera diversa:

```
SUBSCHEMA nome-sub, SCHEMA = nome-sch
ALIAS nome-nuovo1 = nome-vecchio1
ALIAS nome-nuovo2 = nome-vecchio2
.......
REALM nome-area1, nome-area2,.....
SET nome-set1, nome-set2,.........
KEY nome-k1, nome-k2,......
RECORD nome-rec1
 tipo nome-campo10
 tipo nome-campo11
 ..........
RECORD nome-rec2
 tipo nome-campo20
 tipo nome-campo21
 ..........
END
```

Non esiste più, come appunto è proprio del Fortran, una suddivisione in divisioni e sezioni: i nomi del Subschema e dello Schema vengono dichiarati in uno statement iniziale. Per ogni sostituzione di nome si ha uno statement ALIAS. La struttura è descritta dagli statement REALM (elenco di tutte le aree partecipanti a questo Subschema), SET (elenco dei set), KEY (elenco delle chiavi) e RECORD (uno statement per ogni record di Data Base). A proposito di quest'ultimo va detto che il programma Fortran non vedrà i record di Data Base come tali e nella loro intierezza, bensì i singoli campi che li compongono. Subito dopo la dichiarazione del nome record si trova l'elenco dei nomi campi, sempre preceduti da una esplicita dichiarazione di tipo. Ad esempio si potrebbe avere:

```
............
RECORD nome-recI
 INTEGER X
 INTEGER KAPPA
 REAL IPSIL
 CHARACTER *5 Y
 ......
 ....
```

che significa che il record *nome-rec*$_i$ è composto di due campi numerici interi (X e KAPPA), di un campo numerico reale (IPSIL) e di un campo di caratteri (Y).

Si tenga anche presente che il dimensionamento dei campi CHARACTER diviene obbligatorio: Y, nell'esempio, è di cinque caratteri (*5).

Un'altra nota importante: non è permesso l'uso di array quali campi di record (proprio perché ciò non è possibile nello Schema).

Non proseguiamo oltre nell'analisi del Subschema Fortran, solo sottolineiamo ancora una volta che il trattamento è identico, dal punto di vista del sistema, sia per un Subschema Cobol che Fortran e che pure sono identiche, nei due casi, le varie fasi di descrizione e preparazione dei Subschema stessi.

La domanda che sorge spontanea a questo punto è: *quando conviene il Cobol e quando il Fortran?*.

L'esperienza è sicuramente la migliore maestra. Non accontentiamoci, tuttavia, di una risposta così sbrigativa, stabiliamo invece almeno alcuni semplici criteri:

- sicuramente è vantaggioso il Cobol, e così il Subschema Cobol, quando l'applicazione è prevalentemente di carattere gestionale, ovvero il problema principale è quello di trattare l'input/output: ricerca di informazioni, loro conversione, stampa, ecc.

- è vantaggioso il Fortran, e così il Subschema Fortran, quando l'applicazione è prevalentemente di *processo*, ovvero quando si tratta per lo più di eseguire operazioni di calcolo (aritmetico e non), di elaborare algoritmi che non producono grosse moli di informazioni stampate, in una parola quando è il dato ad essere il protagonista principale, nei confronti del volume informativo.

Rimandiamo comunque queste considerazioni ad una sede più adatta. Torniamo al Subschema: una volta scrittane la descrizione in DDL cosa è necessario fare?

Ancora, come per lo Schema DDL e DMCL, esso andrà *tradotto* o *compilato*. Tale attività genererà il **Subschema oggetto**: sarà quest'ultimo la base di partenza per l'applicazione. Di esso si approprierà il programma per strutturare opportunamente la propria memoria: esso *verrà richiamato dal programma* ed il programmatore non dovrà far altro che trattare la parte di Data Base che esso *vede* nè più nè meno come un normale file di I/O.

Prima, tuttavia, di giungere a questo punto è necessaria un'altra attività: una volta generato il Subschema oggetto si rende necessario un controllo, una verifica delle sue strutture e dei suoi formati dati, al fine di garantirne la coerenza rispetto allo Schema cui esso fa riferimento.

Tale attività prende nome di **validazione**. Essa normalmente genera anche vari tipi di informazioni accessorie, sicchè il risultato è costituito spesso dalla scrittura di un secondo file (*control structure*).

Quando non vi siano segnalazioni di errore o di incoerenza (in questo caso andrebbe ripetuta la traduzione del Subschema sorgente) si ottiene una coppia di file oggetto: sono questi ai quali, in effetti, farà riferimento l'applicazione.

Per ognuno dei Subschema che si intende definire sarà necessario ripetere questi passi: alla fine disporremo di una serie, vasta quanto si voglia, di tali coppie, ognuna relativa ad una applicazione, o più spesso ad un loro gruppo.

Non è senza senso chiederci quante di tali coppie definire, o meglio *quanti e quali Subschema utilizzare a fronte di determinate richieste applicative*.

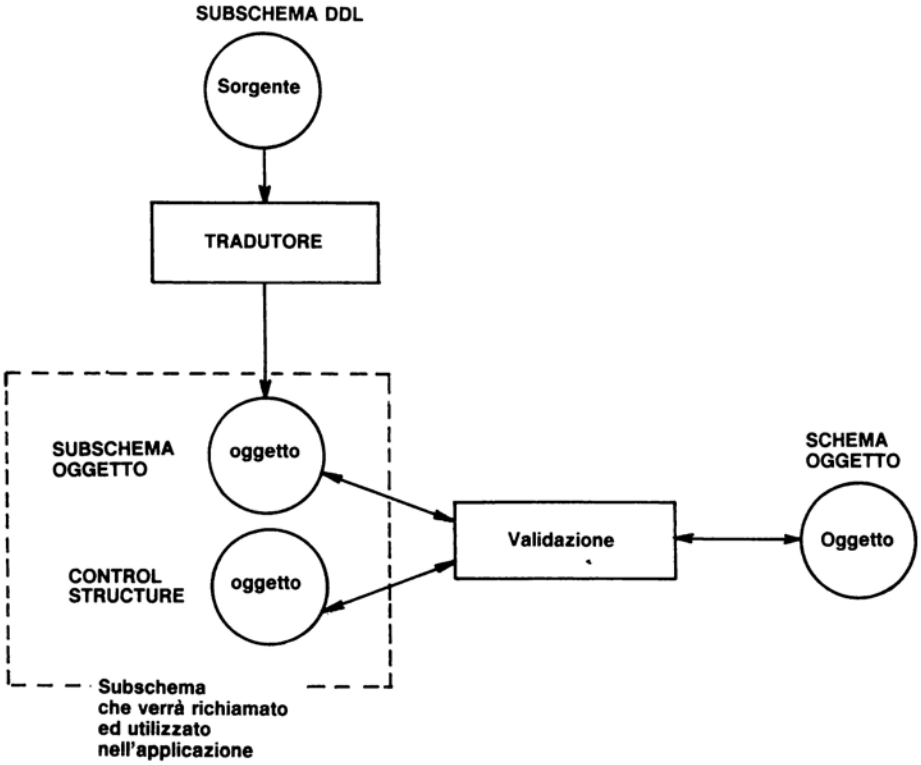


Figura 3.5.a — Il Subschema DDL (sorgente) viene tradotto generando (se non vi sono errori formali) una prima scrittura del Subschema oggetto. In un secondo momento, l'attività di validazione controlla la coerenza allo Schema, facendo riferimento allo Schema oggetto, e genera un file (control structure) contenente delle informazioni accessorie e di controllo. È questa la coppia di file che verrà utilizzata nelle applicazioni.

Per questo argomento, tuttavia rimandiamo all'appendice A in cui la questione viene meglio approfondita e viene descritto e proposto un metodo di scelta. Per ora notiamo che quanto detto, pur se riconosciuto dal Codasyl, non è ciò che avviene nella totalità dei casi, ovvero su tutti i sistemi, ma solo quanto previsto per i software di gestione più raffinati e completi.

Esistono, ad esempio, casi in cui la descrizione di Subschema viene codificata direttamente nel programma, o casi in cui, addirittura, non si ha (sempre nel programma) una visione parziale del tutto, ma ogni elemento risulta a disposizione come se il Subschema coincidesse con lo Schema.

In questi ultimi casi non è tanto importante notare che non disponendo di visioni parziali si perdono le varie possibilità di protezione in esse inserite (fatto già di per sè assai importante), quanto il fatto che una descrizione esterna ed ulteriore, quale quella del Subschema vero e proprio, permette un certo grado di *indipendenza* affatto trascurabile e che ora viene a mancare. Si ricordi quanto detto nei paragrafi introduttivi (1.4): è assai importante poter modificare l'applicazione senza per questo metter mano al Data Base e, viceversa, è assai importante poter modificare il Data Base, od almeno *la sua descrizione*, senza dover modificare (o ricompilare) il programma.

Proprio in quest'osservazione sta il grande vantaggio del Subschema: a volte può bastare modificare tale descrizione (proprio perché esterna) e, dunque, la relativa applicazione secondo le nuove esigenze. Se questa nuova descrizione è ancora coerente allo Schema (e la validazione ce lo dice), il Data Base in quanto tale (lo Schema) può rimanere intatto.

D'altro canto una modifica del Deta Base non è detto che coinvolga una modifica della sua descrizione in termini di Subschema, sempre che quella famosa coerenza sia salva, evitandoci così ricompilazioni di programmi e modifiche sugli stessi.

3.6 Administration

A questo punto si rende sicuramente necessario un riassunto delle attività di realizzazione di un Data Base, o, se si preferisce, delle sue descrizioni a partire dal disegno ed, al contempo la definizione di un nuovo termine: quello di **administration**.

Con tale termine, invalso nell'uso dall'inglese, si intende *quella funzione, affidata ad una o più persone, che ha cura della realizzazione delle descrizioni, del mantenimento del Data Base, del controllo e della supervisione delle applicazioni, dell'effettuazione di eventuali ristrutturazioni ed infine dell'esecuzione delle varie utilities di gestione.*

Procediamo per gradi iniziando a riassumere cosa significhi, realizzazione delle descrizioni in maniere un poco più sintetica e da un punto di vista più generale di quanto fatto fin'ora.

La cosa migliore è sicuramente l'osservazione della figura 3.6.a: in essa sono schematizzati tutti i passi di traduzione dei sorgenti descrittivi analizzati negli scorsi paragrafi.

In teoria la traduzione dei sorgenti di schema DDL e DMCL viene fatta una volta sola, le attività di traduzione e validazione dei Subschema, invece, quante volte serve. Diciamo, in teoria poiché può accadere di ripartire dall'inizio più volte, di dover fare e rifare descrizioni (e non solo a causa di errori, ma anche perchè a volte si procede per tentativi con lo scopo di raggiungere il nostro obiettivo solo dopo aver provato tutte le possibilità).

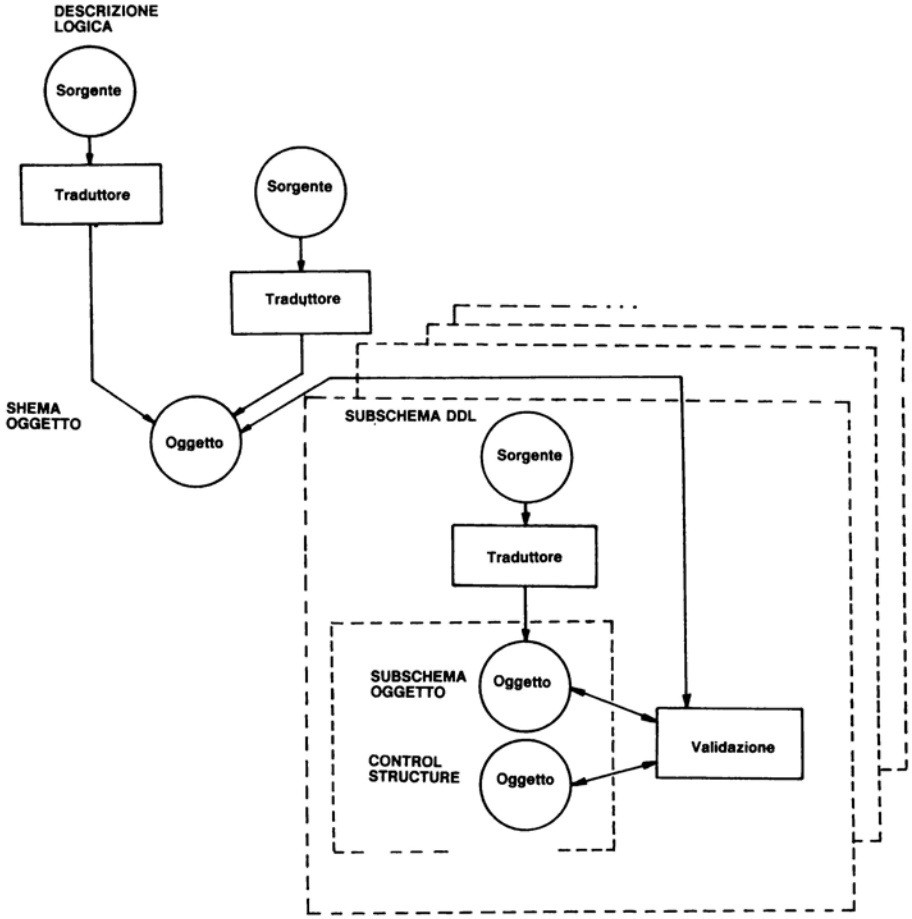


Figura 3.6.a — Le varie fasi preparatorie a cura dell'administrator. Le descrizioni logica e fisica vengono tradotte separatemente e producono la scrittura dello Schema oggetto che rimane unico. Ad esso faranno riferimento tutti i Subschema. Ognuno è costituito solitamente da una coppia di file: il primo, Subschema oggetto, generato dalla traduzione, il secondo dalla validazione, Control Structure.

Lo stesso potrà accadere per il, o per i, Subschema. Alla fine comunque dovremo ottenere una sola descrizione di Schema, lo Schema oggetto, e tutte le descrizioni di Subschema oggetto che ci servono, o che abbiamo deciso di avere, in funzione delle nostre applicazioni.

Il primo compito della funzione di *administration* è proprio questo: passare dal disegno alla realizzazione in oggetto delle descrizioni di Schema, scegliere quali e quanti Subschema saranno da utilizzare, realizzarne le descrizioni oggetto.

A questo punto il Data Base non ci si presenterà più come il solo insieme delle aree (o files) contenenti le informazioni in quanto tali: andranno considerate, unitamente ad esse, le descrizioni di Schema e di Subschema.

Come si può osservare in figura 3.6.b, le vere e proprie informazioni di data Base costituiscono solo una parte di quanto serve alle applicazioni. Esse le utilizzeranno, o vi agiranno, solo attraverso il filtro delle descrizioni.

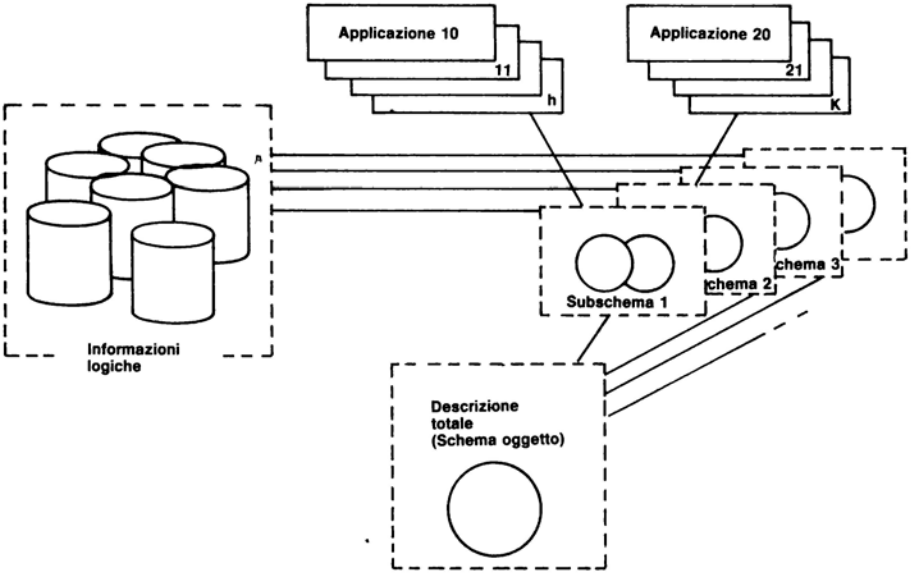


Figura 3.6.b — Il Data Base visto nella sua totalità appare composto di più file descrittivi e delle aree contenenti le varie informazioni logiche. Le applicazioni fanno riferimento a Subschema diversi e possono essere raggruppate proprio in funzione del Subschema che *vedono*. Esse accedono alle informazioni logiche solo *passando attraverso* al Subschema; quest'ultimo è sempre riferito allo Schema che è unico.

Il disegno in figura è forse troppo sintetico, ma l'idea di base, che esso vuol rappresentare, è semplicemente quella che vuole ogni gruppo di applicazioni riferite ad un certo Subschema, che a sua volta riferisce ad un unico Schema; esso agisce da filtro con il Data Base vero e proprio, ovvero con le informazioni effettive.

Gli altri compiti della funzione di administration sono qui solo, purtroppo, accennabili; pur tuttavia vale la pena affrontare alcuni concetti importanti.

Una volta esistenti le descrizioni in formato oggetto, le informazioni di Data Base vanno caricate. Tale fase è estremamente delicata e va seguita con attenzione: l'*administrator* (ovvero chi ricopre la funzione di administration) avrà il compito di stabilire i criteri con cui dovranno essere scritti i programmi di caricamento al fine di ottenere una distribuzione dei record non casuale e di ottimizzare i successivi accessi. La cosa non è semplice e, soprattutto, è strettamente legata al DBMS con cui si lavora. Certamente va tenuto presente che questo primo caricamento deve rispettare quelle scelte strutturali (e di carattere logico) che hanno determinato il disegno.

A questo punto parrebbe che i compiti dell'administrator fossero terminati, ne esistono invece alcuni altri che lo impegneranno costantemente per tutta la vita del nostro Data Base.

Sono questi ultimi, forse, quelli di importanza maggiore.

Si è accennato alla questione del mantenimento ed alla supervisione delle applicazioni. Vediamo brevemente di cosa si tratta, tenendo presente che i due aspetti si integrano in modo completo.

Sono le applicazioni che fanno *crescere* il Data Base in termini di volumi, o che lo *disorganizzano* in termini di ordine dei record (ordine che va mantenuto il più possibile perchè utile ai fini degli accessi): la situazione, è ovvio, richiede un continuo controllo.

Prima di tutto, comunque, è necessaria una certa attenzione alla preparazione delle applicazioni: nè più nè meno come all'atto del caricamento è necessario che ogni applicazione rispetti i criteri di disegno (esso stesso è stato fatto in funzione delle richieste di applicazione).

Controllare le applicazioni è compito dell'administator e significa sostanzialmente che egli deve:

- dare le corrette direttive al programmatore;
- verificare gli aspetti di *privacy*;
- controllare i volumi del Data Base e l'ordine dei record.

Il primo punto riguarda, come già accennato, il fatto che nel programmare l'applicazione si rende necessario rispettare le caratteristiche strutturali e di disegno che sono state date al Data Base. Sarebbe inutile, è ovvio, studiare un disegno opportuno ed adatto a risolvere certe richieste applicative, per poi programmare ricerche di informazioni (o accessi) incongruenti o addirittura contrastanti con tali scelte.

L'administrator dovrà *determinare* in qual modo sia meglio *percorrere* i set, in quali casi si debba accedere direttamente ai record, quando sia preferibile, in definitiva, l'uso di certe strade di accesso al posto di altre.

Un'altra questione importante è quella relativa al secondo punto, quello delle *privacy*: tale termine riguarda *la possibilità di mantenere protette certe informazioni da una lettura, o da un aggiornamento che non si vogliono permettere.*

Abbiamo già visto che il Subschema può utilmente soccorrerci in tale compito.

Esistono altre possibilità (quali l'uso di *Password* (*) di accesso) dipendenti dal DBMS utilizzato. Senza scendere in questi dettagli, va detto che l'administrator dovrà assegnare l'uso di un certo Subschema piuttosto che di un altro e comunicare le opportune *password* di accesso al programmatore e che sarà sua cura predisporre le cose in modo tale da mantenere protette e riservate quelle informazioni che si desidera siano tali e non alla portata di chiunque.

Il terzo punto, quello relativo ai volumi ed all'ordine, è uno dei più delicati.

Già si è accennato al fatto che un Data Base tende a crescere: la continua aggiunta di informazioni non può che avere questo risultato e, di solito, ogni Data Base ha, per sua stessa definizione e finalità, il preciso compito di accettare informazioni sempre nuove che, anche se non ridondanti, si aggiungono alle vecchie che non devono andare perdute.

Tutto ciò non può che portare a problemi di gigantismo non sempre accettabili (il Data Base ha quasi sempre anche un aspetto *storico* e le informazioni possono dovervi vivere per diversi anni). Certamente, all'atto del disegno, si sarà dovuto tener conto del fatto che esista una certa frequenza di cancellazioni e conseguente nuova disponibilità di spazio strutturando, così, set e record in maniera opportuna. Altrettanto certamente la crescita risulterà spesso inevitabile e dunque dovrà essere soggetta ad un controllo preciso.

La questione non è solo relativa al volume totale: una crescita disordinata può portare i record qua e là, può lasciare spazi vuoti (inutilizzati ed inutilizzabili), può, soprattutto, provocare una perdita in termini di facilità di accesso.

La struttura studiata inizialmente doveva tener conto del raggruppamento dei record e della relativa facilità di accesso: se un alto numero di aggiornamenti (in termini di nuovi inserimenti, cancellazioni e modifiche) provocasse disordine, andrebbe perso proprio quel vantaggio strutturale e di disegno che voleva i record disposti nel modo prevedibilmente migliore. Ancora il discorso si fa complesso perché intimamente legato al DBMS particolare; è certo, tuttavia, e facilmente comprensibile, che l'ordine in cui i record sono presenti deve essere corrispondente al disegno: è preferibile che in una stessa pagina si trovino l'owner ed i suoi member, piuttosto

(*) Password, o parola d'ordine, sta per parola chiave, o frase, da *pronunciare prima che la funzione richiesta venga eseguita*; sarà compito del DBMS controllare che l'enunciazione di tale parola sia corretta prima di permettere che quella funzione divenga eseguibile.

che l'owner in una pagina, il primo member in un'altra, il secondo in un'altra ancora e così via.

Questa situazione porterebbe ad un notevole moltiplicarsi degli accessi.

Qual è, allora, il compito dell'administrator?

In sostanza controllare l'ordine/disordine che si viene creando a fronte dei (sicuramente) numerosi aggiornamenti cui il Data Base è soggetto: dopo un certo periodo, e dopo un certo numero di aggiornamenti, potrà essere necessario *riordinare*, ovvero *scaricare* tutto su di un file sequenziale e *ricaricare* con un opportuno programma. Questa operazione di scaricamento e ricaricamento può anche essere prevista dal DBMS, ma, e questa è la questione più importante, deve trattarsi di ricaricamento logico e questo nessun DBMS è in grado di prevederlo compiutamente proprio perchè il ricaricamento logico può prevederlo proprio perché è funzione delle strutture particolari, ossia del disegno. Di questo deve insomma farsi carico l'administrator: egli deve predisporre un metodo che permetta di riportare i record nella loro posizione ottimale, anche se, per vari motivi di aggiornamento, sono finiti fuori posto, e ciò non può farlo se non indicando i criteri di un programma di *scaricamento* e di uno di *ricaricamento*.

Come può accorgersi, l'administrator, del momento in cui tale operazione diviene necessaria?

Certo non solo da una crescita abnorme dei volumi: sarebbe troppo tardi. Per sua fortuna ogni DBMS prevede delle attività a sé stanti che possono essere lanciate separatamente ed indipendentemente da ogni applicazione, che gli forniranno le necessarie indicazioni. Si tratta delle *Utilities*, più o meno numerose ed efficaci sui diversi sistemi, e più o meno di facile utilizzo.

Esse non servono solo quale *misura* del Data Base, spesso hanno anche altre funzioni e proprio per questo meritano un discorso a sè stante.

Va tenuto presente anche l'altro dei due grossi problemi cui abbiamo accennato e che l'administrator si trova a dover affrontare già in una fase iniziale: quello della scelta dei Subschema.

Per quest'ultimo rimandiamo nuovamente all'appendice A in cui è proposto un metodo di scelta di una certa utilità.

3.7. Il Data Manipulation Language

Abbiamo visto come il Subschema funga da *filtro* tra Data Base e procedura applicativa e anche come esso sia, per formati ma anche caratteristiche generali, strettamente aderente al tipo di linguaggio applicativo stesso. Ci restano da vedere, dunque, le applicazioni vere e proprie, ovvero i programmi, ed in particolare le novità riguardanti il linguaggio in cui sono scritte. Vedremo tra l'altro come il Cobol ed il Fortran, ad esempio, vadano integrati da verbi di tipo particolare: quelli che permetteranno gli accessi al Data Base.

Prima di tutto, va notato che tali verbi aggiuntivi costituiscono un vero e proprio linguaggio che prende nome **DML (Data Manipulation Language)**.

Esso è detto solitamente *host language dependent*, il che significa, letteralmente, che esso è, dipendente dal linguaggio ospitante e ciò proprio nel senso che si è appena accennato e che riguarda il formalismo di scrittura, le caratteristiche funzionali, la tipologia dei dati. Più genericamente tale denominazione implica anche, però, la dipendenza dal compilatore per quanto riguarda la traduzione in oggetto di tali verbi: essi vengono accettati come una estensione del linguaggio (Cobol o Fortran che sia) ed il relativo compilatore li traduce in richiami alle routine di accesso al Data Base senza che venga espressa alcuna richiesta supplementare da parte dell'utente.

Si è detto solitamente, perché non tutti i sistemi presentano questa caratteristica: alcuni richiedono un vero e proprio *cambio di linguaggio* all'interno del programma (il che corrisponde al richiamo di un diverso, appropriato compilatore per quel DML), altri ammettono che le applicazioni vengano scritte solo in linguaggi ad hoc e sostanzialmente costituiti da soli verbi DML, più pochi altri di trattamento, l'effetto di tali procedure verrà poi passato ai programmi normali.

Queste sono vie, comunque, per lo più obsolete (*) e la tendenza attuale è quella dell'*host language dependent*. Già accettata dal Codasyl, essa sicuramente subirà evoluzioni, ma altrettanto sicuramente non si discosterà sensibilmente dal criterio di base.

Vediamo ora quali sono le principali funzionalità ammesse dal DML standard Codasyl tenendo presente che, al solito, ogni particolare sistema può presentare qualche differenza. Qui di seguito esaminiamone rapidamente un elenco:

apertura	READY
accesso	FIND GET
scrittura	STORE
modifica	MODIFY
cancellazione	ERASE
connessione in un set	CONNECT
sconnessione da un set	DISCONNECT
chiusura	FINISH

(*) Tuttavia si stanno sviluppando nuovi linguaggi *ad hoc* con caratteristiche assai innovative: se ne parlerà in un prossimo paragrafo.

abbiamo posto, di fianco alla denominazione della funzione il verbo corrispondente. Esso avrà poi un suo formato specifico di scrittura corrispondente al modo in cui la funzione deve essere svolta: ad esempio si potrà voler accedere direttamente ad un record, ed allora si potrà scrivere qualcosa come:

FIND ANY *rec-n*,

oppure per scorrimento del set, ed allora si potrà scrivere qualcosa come:

FIND NEXT *rec-n* WITHIN *set-m*.

Inoltre esistono differenze relative al linguaggio dal quale il DML dipende. Ancora per fare un esempio:

FIND NEXT *rec-n* WITHIN *set-m*

è scritto in DML Cobol, mentre:

FIND (NEXT, RECORD = *rec-n*, SET = *set-m*)

è scritto in DML Fortran.

I formati particolari, ancora, sono strettamente dipendenti dal sistema specifico e dunque non li tratteremo; analizziamo invece ognuno dei verbi elencati cercando di spiegare più in dettaglio il significato di ogni funzione:

READY — permette l'apertura dei file di Data Base, ovvero delle *aree*, anzi, meglio, delle *aree* che il Subschema usato nella particolare applicazione *vede*; tale apertura può riguardare anche una sola di queste aree, o più, o tutte; è permesso aprire in RETRIEVAL per sola lettura, in UPDATE per lettura e scrittura, od in LOAD per sola scrittura;

FIND — è il verbo che permette l'accesso fisico; nel formato specifico di scrittura verrà definito il criterio di ricerca e selezione del record voluto, una volta che il sistema lo avrà individuato, porterà tutta la pagina fisica di Data Base che lo contiene, nel *buffer* di memoria del programma; va posta attenzione al fatto che il record non è, a questo punto, elaborabile, sarà prima necessario portarlo in area di lavoro;

GET — provvede a portare in area di lavoro il record richiesto (e segue, dunque, la FIND); normalmente si parla di UWA (User Work Area) denominando così la zona di memoria del programma in cui sono previste le *immagini record* del Subschema e sulle quali procederà l'eleborazione;

STORE — permette la scrittura del record, i suoi campi dovranno essere inizializzati, andrà definita la posizione (sotto quali eventuali owner...) e quindi verrà fatta la richiesta di scrittura;

MODIFY — è la richiesta di modifica; essa può essere *logica*: si inizializzano i campi con i nuovi valori e si fa la richiesta di modifica; oppure *di appartenenza*: si definiscono gli owner sotto i quali il record dovrà disporsi e la modifica consisterà nella riscrittura, da parte del sistema, dei pointer;

ERASE — è la richiesta di cancellazione; essa può essere *fisica*: il record viene cancellato effettivamente, ovvero i pointer dei record collegati a quello da cancellare vengono aggiornati in modo da ripristinare eventuali collegamenti nei set, inoltre loé spazio precedentemente occupato viene reso nuovamente disponibile; oppure può essere *logica*: il record rimane fisicamente presente, ma è reso illeggibile; tale strada è di solito seguita nelle applicazioni in cui è importante il risparmio di tempo (ad esempio è il caso degli aggiornamenti in tempo reale) poiché tale cancellazione risulta assai più rapida, tuttavia sarà necessario periodicamente *ripulire* (esisterà una opportuna *Utility* per farlo) le pagine di Data Base dai record presenti, ma ormai inutili;

CONNECT — permette di connettere record già presenti nel Data Base ad un owner in un set di tipo opzionale; vale ovviamente per i soli sistemi che ammettono l'esistenza di tale tipo di set; è sufficiente definire quale debba essere l'owner, prima di richiedere la connessione;

DISCONNECT — ancora vale solo per i sistemi che ammettono l'esistenza di set opzionali e serve per *staccare* il record dal suo owner in quel set;

FINISH — permette di chiudere le aree aperte con il verbo READY; è possibile chiuderne solo alcune o tutte.

3.8 Utilities

Si è già notato come col termine **Utilities** si designano certe attività, specifiche di ogni DBMS, automatiche e del tutto indipendenti dalle applicazioni vere e proprie. Possiamo precisare che ogni DBMS presenta certe funzioni di utilità comune, che sono necessarie ad una corretta gestione amministrativa del Data Base.

Cosa significa ciò? In sostanza, semplicemente, che l'attività di administration è più o meno facilitata da una serie di moduli (o programmi) predisposti opportunamente ed atti a svolgere quei compiti di risistemazione dei dati, riordino degli spazi, analisi dello stato fisico ecc... che normalmente devono essere svolti dall'administrator con una certa frequenza.

Le *Utilities* ovviamente non riguardano i soli aspetti di riordino od ottimizzazione degli spazi: a volte esse sono relative a ristrutturazioni (addirittura dello Schema), a volte rivestono carattere di necessità (come quelle di inizializzazione, nei DBMS che richiedono tale fase preliminare), a volte riguardano il caricamento dei dati, ecc.

Come si vede il panorama è piuttosto vasto ed ancora una volta, a questo proposito, è bene ricordare che il sistema specifico, più o meno evoluto, o più o meno sofisticato permette un numero più o meno alto di tali funzionalità automatiche. Anzi è invalso nella tradizione professionale, valutare un DBMS proprio in funzione del numero e delle capacità delle *Utilities* di cui dispone. Ciò non è del tuto corretto poiché l'efficacia del sistema non può che dipendere, in primo luogo, dalle sue capacità logico-fisiche, tuttavia sarà sempre bene tenere presente che le funzionalità non previste come automatiche dovranno essere *programmate* dall'administrator, e che dunque, la deficenza di qualcuna delle *Utilities* può rappresentare uno svantaggio non da poco.

Procediamo comunque con ordine, domandandoci, innanzitutto quali sono le funzionalità di gestione ricorrenti alle quali l'administrator deve prestare primariamente attenzione.

Forse la questione più importante è quella dell'ordine e dello spazio: si è già detto che un certo numero di aggiornamenti non può che provocare disordine e che questo può essere assai dannoso in termini di allungamento dei tempi di accesso. Ciò può accadere, ad esempio, a causa di cancellazioni: certe cancellazioni logiche dei record possono lasciare nel Data Base dei set *lunghi* da percorrere pur accedendo a pochi record *vivi*. Ecco allora una prima necessità: ripulire periodicamente dai record logicamente cancellati e ristabilire set di soli record *vivi* ricompattando lo spazio: ciò è fatto dalle Utilities dette di **delete**.

Altra causa di disordine può essere rappresentata da un eccessivo riempimento: quando un record va a disporsi nella stessa pagina del suo owner, l'accesso, lo si è già detto altrove è facilitato. Se esso non trova spazio nella pagina, invece, si generano set che per essere percorsi richiedono accessi fisici molteplici. Ecco allora che una Utility di **scaricamento** su file sequenziale e **ricaricamento** logico (nel senso di ricaricamento secondo le priorità owner-suoi-member prescelte) da quel file può rappresentare un grosso vantaggio.

Non sono molti i tipi di DBMS che dispongono di tale tipo di Utilities. Di fatto i criteri di priorità che vanno scelti per il ricaricamento costituiscono un problema di una certa delicatezza e che richiede un'attenzione del tutto *umana*!

Proprio per questo come si è notato nel paragrafo 3.6 lo scaricamento e ricaricamento di un'area integrata è consigliabile venga fatto più spesso da programmi studiati opportunamente che non da Utilities predisposte. Comunque vadano le cose, tuttavia, è necessario che l'administrator sappia quando è il momento di scaricare e ricaricare: a questo compito provvedono varie Utilities di **analisi**. Tra queste sono da segnalare almeno quelle di cosiddetta **analisi statistica** e quelle ad **analisi di dettaglio**".

Le prime generano solitamente delle report quali quella di figura 3.8.a, ovvero degli istogrammi (da qui il nome dato al tipo, *statistico*, anche se in senso assai lato) dai quali è possibile ricavare due indicazioni:

a) quanto spazio per pagina (o per gruppo di pagine) è stato percentualmente utilizzato;

b) quanti dei record ammessi per pagina (o per gruppo di pagine) sono stati percentualmente inseriti, o, che è lo stesso qual'è la percentuale di utilizzo delle DBK.

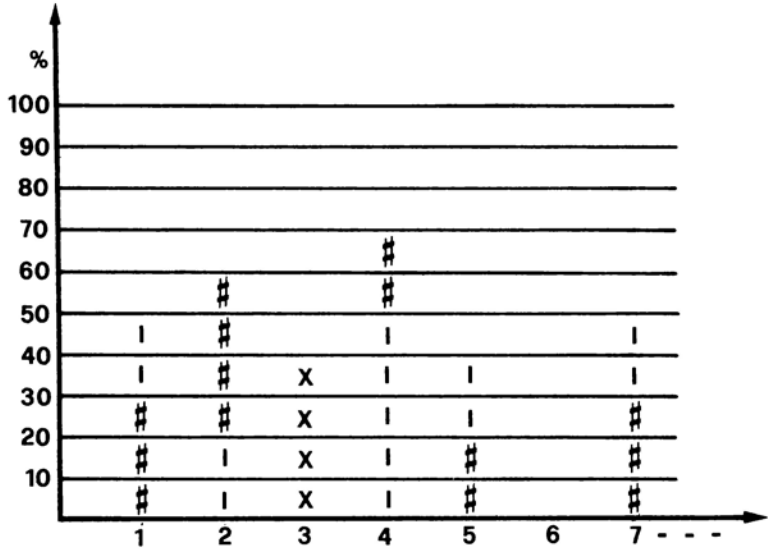


Figura 3.8.a — Istogramma delle occupazioni di pagina: in ascisse è indicato il numero di pagina, in ordinate la percentuale di riempimento o di utilizzo di DBK. Il simbolo # indica la percentuale di spazio, il trattino I la percentuale di utilizzo di DBK per pagina. Così: pag. 1 ha un'occupazione di spazio del 30% e un utilizzo di DBK del 50%, pag. 2 è piena al 60% ma con solo un 20% delle DBK utilizzate, pag. 3 ha un uguale riempimento e utilizzo di DBK del 40% (per questo, il simbolo X), pag. 6 è vuota ecc.

Per ben comprendere il senso di queste informazioni si osservi bene l'esempio di figura 3.8.a: in pagina 1 lo spazio è stato utilizzato al 30% e le DBK al 40%, in pagina 2 lo spazio al 60% e le DBK al 20% e così via. L'importanza di queste valutazioni percentuali è notevole: prima di tutto si ottiene una visione d'assieme dello stato del nostro Data Base, in secondo luogo si scoprono facilmente le pagine troppo vuote o troppo piene ecc...

Un altro fatto importante è saper riconoscere tra distribuzioni di vario tipo quali rispecchino situazioni accettabili e quali altre no; si esaminino i casi di figura 3.8.b: l'esempio (1) corrisponde ad una situazione di avanzato riempimento, ma non ancora critica, infatti esistono varie pagine con sufficiente spazio libero e ben poche sono quelle in cui non potrà andare a disporsi alcun record (la 5 che non ha più spazio, la 7 che non ha più DBK disponibili).

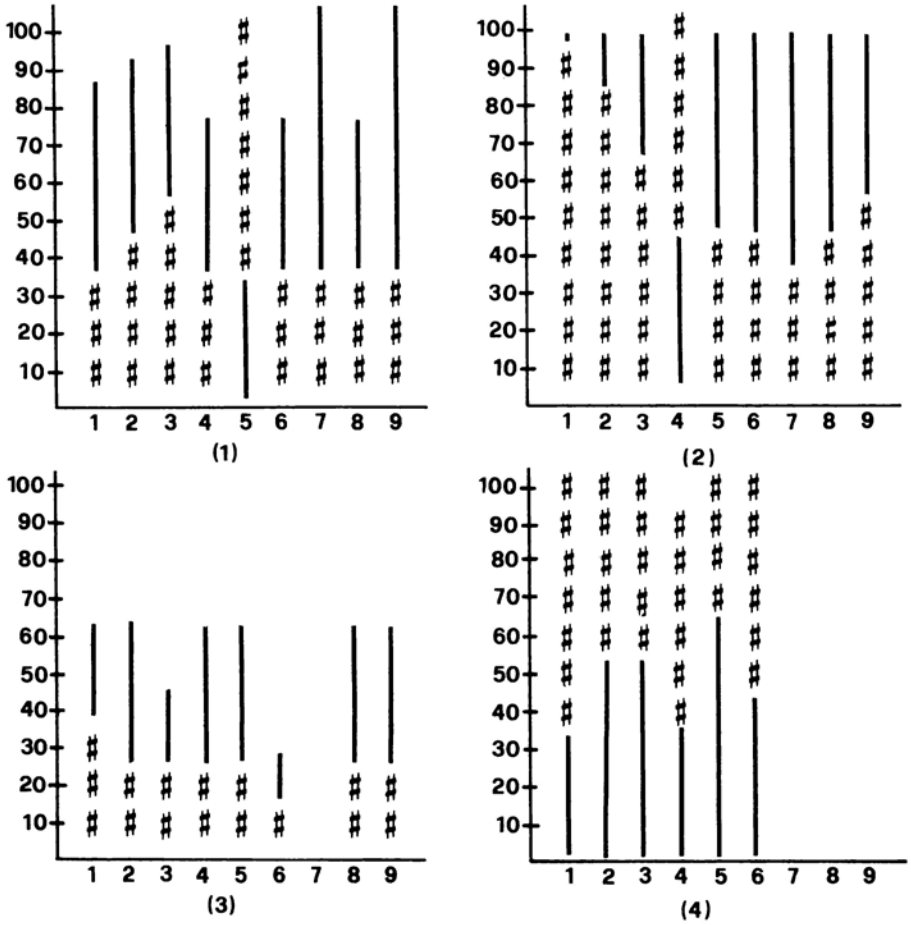


Figura 3.8.b — Vari casi di occupazione percentuale dello spazio (ancora simbolo #) e di uso delle DBK (simbolo I).

Ben diverso l'esempio (2): la situazione è sicuramente critica non essendovi praticamente più alcuna possibilità di inserimento: bisognerà scaricare e ricaricare o addirittura modificare il Data Base aumentando le dimensioni di pagina.

Non del tutto soddisfacente neppure il caso (3) a meno che esso non rispecchi la situazione iniziale, ovvero il fatto che fino a quel momento è stato fatto un basso volume di aggiornamenti. Se così non fosse vorrebbe dire che abbiamo sbagliato le dimensioni di pagina.

L'ultimo caso, il (4), poi è il peggiore di tutti: abbiamo previsto troppo pochi record per pagina, così pur essendoci spazio, non abbiamo comunque possibilità di fare inserimenti e dobbiamo obbligatoriamente operare una modifica.

Valeva la pena esaminare tutti questi esempi un po' più in dettaglio per un motivo assai semplice: alcuni DBMS danno la possibilità di lanciare queste Utilities in fase di simulazione: si definisce il numero dei record da caricare e la loro grandezza, si stabiliscono dei parametri che rispecchino le caratteristiche del caricamento e si lancia l'Utility prima di eseguire il caricamento stesso ottenendo quelle indicazioni che, una volta paragonate tra loro, potranno guidarci verso la soluzione migliore.

Indicazioni ancora sull'utilizzo dello spazio e delle DBK son date in modo più approfondito nell'altro tipo di Utilities che abbiamo chiamato di analisi di dettaglio. Ne è presente un esempio in figura 3.8.c: come si vede, questa volta si sa esattamente quanto spazio è disponibile e quante DBK in ogni pagina (o gruppo di pagine).

PAGE	USED	
	DBK	SP
1	3	436
2	6	812
3	28	824
4	1	120
5	6	80
...	...	...

Figura 3.8.c — Esempio di tabella del riempimento: in pagina 1 sono presenti 3 record (3 DBK utilizzate) con un'occupazione di 436 bytes (scegliamo questa come unità di misura, ma potrebbe trattarsi di parole o qualunque altra unità ammessa dal DBMS); in pagina 2 vi sono 6 record con una occupazione pari a 812 bytes ecc.. Si noti come risultino evidenti casi più o meno particolari come quello di pagina 5: vi sono ben 6 record, ma evidentemente di dimensioni minime, occupano, infatti, solo 80 bytes.

Un ultimo tipo di utilities di cui va dato un cenno sono quelle di **dump**, ovvero di stampa fisica delle pagine: esse servono per produrre in stampa l'immagine completa della pagina, o delle pagine richieste in modo non interpretato (ossia in ottale o esadecimale).

Sono utili, ovviamente per una verifica di dettaglio assai avanzata e sono assai specifiche del sistema.

3.9 Le modalità operative tradizionali ed ad-hoc

Le modalità operative cui abitualmente si può ricorrere quando si disponga di un Data Base sono di vario tipo, dalle più tradizionali a quelle recentemente nate con l'evoluzione dei prodotti; alcune di queste sicuramente peculiari ed utilissime. Tutto fa prevedere che in un futuro non lontano ve ne saranno di ancor più numerose; forse la via è segnata e tra non molto praticamente tutto il lavoro verrà svolto attraverso funzionalità nuove e di facile utilizzo.

Esaminiamo tuttavia lo stato dell'arte e partiamo con una rapidissima analisi proprio di quelle funzionalità che abbiamo detto tradizionali. Il termine potrebbe trarre in inganno: la storia del Data Base è assai breve e così quella del Data Processing in generale: con modalità operative *tradizionali* si intendeva solo indicare quelle che già esistevano prima della nascita del Data Base in quanto tale o che con esso, più o meno contemporaneamente, sono nate. Ci riferiamo alla modalità operativa specificatamente **batch** ed a quella di **tempo reale**.

Si tratta sicuramente di termini conosciuti. Ricordiamo comunque che con termine *batch* gli anglosassoni designano la pala del fornaio: in questo modo di lavorare i dati sono forniti in input ai programmi, per così dire, a... palettate. Più precisamente: l'archivio viene aggiornato a periodi prestabiliti, una volta raccolta una certa quantità di dati d'aggiornamento viene lanciato il programma con quei dati in input. Ne segue che l'archivio non è quasi mai realmente *in linea* con i fatti, ovvero esso non è detto che rispecchi la reale situazione del momento se non subito dopo gli aggiornamenti stessi (o magari neppure questo, perché già nel momento in cui si è effettuato il lancio degli aggiornamenti la situazione è andata cambiando).

Questo ora detto è il punto più importante e per il quale abbiamo comunque voluto spendere due parole sul *batch*: raramente avviene che un'applicazione di questo tipo sia lanciata con una frequenza più che giornaliera (ed è giusto che sia così) e dunque non ha senso utilizzarla per mantenere nell'archivio l'immagine della situazione reale momento per momento. Quando allora è utile? Come è facile immaginare, quando il problema è di per sua natura periodico, ma ancor più quando abbiamo a che fare con stampe voluminose, con ricerche di informazioni lunghe, complicate e laboriose, quando, insomma, il tempo che ci vorrà perchè i programmi vengano eseguiti, è la minore delle preoccupazioni.

Si badi: la mancanza di allineamento con la realtà non è detto sia così fastidiosa come potrebbe sembrare a prima vista: la banca che a fine mese deve stampare i bollettini riassuntivi dei movimenti per ogni conto corrente, non può che partire ad esaminare la situazione da una data fissata ben precisa, ad esempio il 30 o il 31 di ogni mese, ed ogni movimento che avvenga il giorno dopo magari durante l'esecuzione non potrà che comparire nel bollettino successivo. D'altro canto proprio questo è un caso in cui l'applicazione non può che essere *batch* anche per altri motivi: i bollettini sono numerosissimi, e dunque la loro stampa non può che essere lunga, va risalita la storia di tutto un mese, e dunque il numero di accessi ai record può essere elevato producendo esecuzioni più o meno lunghe, ecc.

Detto ciò come si prepara un programma batch? Come si sarà già capito, si tratta semplicemente di scrivere programmi in un linguaggio evoluto facendo uso dei verbi DML che abbiamo visto nel paragrafo 3.7. Non è detto esistano i soli linguaggi che in quell'occasione abbiamo esaminato (vale a dire i DML del COBOL e del FORTRAN), ogni sistema potrebbe averne uno suo particolare, sicuramente però, quelli sono attualmente i più diffusi.

Un'ulteriore precisazione: spesso si dice che si lavora in batch anche quando si fa uso di funzionalità particolari di interrogazione, tipo quelle di **query** che vedremo tra breve, o di aggiornamento; dal punto di vista del lavoro di sistema ed allineamento dell'archivio con la realtà, ciò non è del tutto sbagliato; tuttavia noi abbiamo voluto raggruppare queste ultime funzionalità sotto una loro separata denominazione: **ad-hoc**.

Passiamo ora ad esaminare, ancora assai rapidamente, alcuni concetti base di *tempo reale*: esso è quel modo di operare che permette di mantenere in linea l'archivio. Non appena la situazione cambia, la modifica viene riportata sull'archivio stesso, gli aggiornamenti non avvengono a periodi prefissati, ma la loro frequenza è proprio quella stessa dei cambiamenti che la realtà ci propone. Ciò è realizzabile per la grande diffusione che hanno oggi avuto le reti di trasmissione dati, o, quanto meno, i terminali comunque collegati al sistema centrale che gestisce l'archivio.

Si pensi al caso della banca: contrariamente a quanto avveniva per i bollettini, è assai importante che i movimenti (versamenti, prelievi, ecc.) vengano immediatamente riportati sull'archivio. Quello che ci accade ogni volta che andiamo ad uno sportello di banca, di vedere cioè l'impiegato che registra sul terminale la nostra operazione e ci fornisce immediatamente il saldo attuale (e *vero*), è forse l'esempio più semplice e significativo dell'uso del *tempo reale*.

Vale la pena vedere, almeno in linee assai generali, come è tecnicamente realizzabile questa modalità operativa. Di solito le case costruttrici forniscono un sfotware di gestione che permette di realizzare il cosiddetto **ambiente transazionale** (vedremo tra breve di precisare meglio il significato di tali termini) che può essere separato o integrato nel DMBS, ma che deve comunque essere in grado di colloquiare con esso. Tale software prende genericamente nome di **software transazionale** ed è composto praticamente sempre nel seguente modo:

a) dispone di un modulo, o nucleo centrale, che fa da supervisore e controllore di tutti gli altri moduli;

b) dispone di uno o più moduli che hanno il compito di interagire (ricevere o inviare ordini) con il sistema operativo;

c) dispone di uno o più moduli che permettono il colloquio con la rete, ovvero con i terminali, in definitiva con lo stesso utente;

d) dispone di uno o più moduli in grado di colloquiare con il DBMS e dunque di provvedere agli accessi al Data Base;

e) dispone di uno o più moduli in grado di mandare in esecuzione programmi utente previamente inseriti in un'apposita libreria.

In quasi tutti i casi, almeno i moduli di cui al punto e) sono ovviamente altamente personalizzabili, spesso lo sono anche altri moduli, mentre in alcuni prodotti il tutto è fornito come una scatola chiusa ed il solo intervento permesso all'utente è quello di scrivere i propri programmi applicativi.

Tali programmi devono avere come loro principale caratteristica la brevità, saranno spesso conversazionali, dovranno prevedere il minor numero possibile di accessi fisici (il terminalista non deve e non può aspettare la risposta ad una sua richiesta per un tempo superiore ai pochi secondi). Si sceglie una funzione applicativa, si codificano uno o più programmi che richiamandosi l'un l'altro la svolgono e si è realizzata una **transazione**. Nella maggior parte dei casi il programma applicativo è scritto in COBOL e comunque farà uso di opportuni verbi DML.

Il software transazionale può venir normalmente tenuto nella memoria del sistema per tutto il tempo desiderato: è come un programma che vien lanciato ad un determinato momento e resta in esecuzione perpetua fino a quando non lo interrompiamo esplicitamente. Durante questo periodo potrà essere lanciato da terminale il comando di innesto di una certa transazione, i vari moduli provvederanno al richiamo dei relativi programmi ed alla loro esecuzione. Magari nel frattempo da un altro terminale potrà essere stato inviato analogo comando per un'altra o addirittura per la stessa transazione: nulla di problematico, di solito le code di attesa si creano solo quando due transazioni vogliono accedere contemporaneamente in scrittura sulla stessa pagina di Data Base.

A questo punto il discorso si fa assai delicato essendo la cosa strettamente dipendente dallo specifico sistema: ci serve comunque ancora una volta ad evidenziare come in questa modalità operativa sia assai importante, anzi fondamentale, porre la massima attenzione nel cercare di minimizzare il numero di accessi fisici.

Si è detto, poco fa, dell'esistenza di altre modalità operative, forse di nascita più recente, che abbiamo voluto raggruppare sotto la denominazione **ad-hoc**: si tratta di modalità che fanno uso di un loro linguaggio, che utilizzano a volte un loro DML diverso da quello standard, che hanno spesso carattere interattivo, che, soprattutto, nascono con intenti ben precisi, per risolvere problemi specifici.

Una loro classificazione è assai difficile poichè dipendono dai particolari DBMS, tuttavia tutte hanno almeno la seguente caratteristica: *servono ad esaudire richieste estemporanee*.

Che accadrebbe se in fase di analisi ci si fosse dimenticati di una particolare funzione? E se la necessità fosse nata solo in un secondo momento, per così dire, a cose fatte? Avremmo solo la scelta di codificare un programma batch (e la cosa può risultare lunga e non adatta), o di ripersonalizzare da capo l'ambiente transazionale per inserirvi una nuova transazione (cosa probabilmente ancora più lunga). Ecco allora la necessità di altre modalità operative che permettano di risolvere problemi più o meno complessi in tempi brevi.

Tra queste spiccano per importanza quelle di *query*, interattivo o meno, che tuttavia, poiché strettamente legate all'ambiente *end-user*, non possono che venir trattate a parte e solo dopo aver precisato quest'ultimo concetto.

3.10 Dalle modalità operative end user ai linguaggi di IV generazione

La più recente evoluzione di tutti i vari tipi di software sembra essere avvenuta all'insegna della maggior attenzione possibile per l'**end-user** e tutto fa presagire che in futuro tale tendenza sarà ancora più accentuata. Cos'è questa figura di utente finale (*end-user* appunto) di cui da più parti si sente parlare? Sostanzialmente si tratta della persona non esperta in data processing che utilizza però le applicazioni preparate per lui o da lui stesso richieste, che ha bisogno delle informazioni presenti nel Data-Base, che trae giovamento, nel proprio lavoro, dall'uso, in definitiva, dell'elaboratore. Si tratta dell'amministrativo, del contabile, della segreteria come del manager. Essi non possono che essere tutti agevolati dal disporre di un terminale: non è più sufficiente oggi ricevere i tabulati, è più rapido leggere l'informazione necessaria sul video di un terminale. Non è pensabile dover ricorrere al programmatore ogni volta, magari per esigenze nate all'ultimo momento e che richiedano soluzioni veloci.

Se da un lato con ciò torniamo al discorso delle esigenze estemporanee, dall'altro siamo costretti a chiederci come possa una persona con quelle caratteristiche e cultura utilizzare il terminale. Per fortuna l'atteggiamento verso l'informatica è andato cambiando e tutti oggi vorrebbero un terminale od un personal-computer (ed a questo proposito, tra breve, ci sarà qualcos'altro da dire) sulla propria scrivania, ma come servirsene? Non deve essere loro necessario sapere come è fatto un elaboratore, come si scrivono i programmi, neppure come è fatto e disegnato il Data Base.

La risposta non può che stare in quelle modalità *ad-hoc*. Esaminiamo le principali caratteristiche di quelle di **query**. Esse sono composte di un numero limitatissimo di comandi che tutti possono imparare e che permettono rapide interrogazioni. Di soli-

to agiscono attraverso un *filtro*, realizzato invece da personale esperto DP, in grado di pre-aggregare certe informazioni. All'utente finale viene fornito un elenco contenente la descrizione delle informazioni volute e con un semplice comando egli può richiamarle sul video.

Questo insieme di informazioni può essere visto come un vero e proprio record. Esso, tuttavia, non esiste in quanto tale nel Data Base. Le informazioni che lo compongono sono state prese, per così dire, qua e là... e dunque, proprio per questo viene detto **record virtuale**: esso non è altro che *l'aggregazione, predisposta dall'esperto DP, di più campi di record diversi, scelti secondo certi ben definiti criteri.* Si immaginino tre record A, B, C composti di tre campi ognuno. Si potrebbe parlare di *record virtuale* per l'informazione A1-B1-C3 contenuta nel primo campo di A, nel primo di B e nel terzo di C. L'*end-user* disporrà di un elenco di tutti i record virtuali esistenti con relativa indicazione del tipo di informazione ivi contenuta. Ammettiamo gli serva proprio quello del nostro esempio, potrà scrivere cose come

TROVA A1 - B1 - C3

DOVE A1 È "XXX"

ottenendo un elenco di tutti i contenuti dei campi A1 B1 e C3 con XXX come contenuto di A1.

Già, addirittura in italiano: molte, se non tutte tra queste funzionalità vengono fornite nella lingua nazionale o sono traducibili.

Molto interessante tra le modalità di *query* una recentissima evoluzione: su certi sistemi il concetto di "*filtro*" è servito a cambiare la visibilità dal Data Base da reticolare a relazionale. È come se al posto del record virtuale venissero utilizzate le tabelle del relazionale di più immediata comprensione e facile uso da parte dell'end-user. Ciò permette di simulare in tutto e per tutto le strutture ed i principii del relazionale sia nelle fasi di interrogazione pura e semplice, accedendo a tabelle della più ampia varietà aggregativa, sia di produrre stampe (semplici o complesse).

Le funzionalità di query non si fermano a quelle elencate, anzi non si fermano neppure all'end-user. Si è accennato prima al fatto che possono essere interattive: alcuni DBMS permettono addirittura di utilizzare i verbi DML standard per lavorare in colloquio diretto con il Data Base via terminale; altri hanno un loro linguaggio specifico per ottenere lo stesso scopo. In questi casi, però, difficilmente si può parlare di orientamento end-user.

Altrettanto dicasi per quelle funzionalità *ad-hoc* che oltre al query (o interrogazione) permettono anche aggiornamenti, modifiche, reporting strutturato.

Ancora una volta il discorso si fa troppo particolare, torniamo, invece, all'end-user per notare come normalmente egli preferisca avere sulla scrivania un personal computer piuttosto che un terminale. Il perché è presto detto: il primo gli permette di fare dell'altro, oltre al query, in maniera facile ed immediata. Non è detto che il terminale, strettamente legato al sistema com'è, possa fare altrettanto. Ecco perché si

tende a rendere il personal computer compatibile o collegabile con i sistemi più grandi e tradizionali ed ecco perché aumenta di importanza il Data Base distribuito: si potrebbe tenere su ogni personal computer collegato, una copia di quella parte di Data Base che interessa quel particolare utente e solo quella, aggiornandola periodicamente. Ciò presenterebbe il vantaggio di proteggere ulteriormente il Data Base centrale e di facilitare ancor più l'uso al non esperto.

A buon diritto in questo discorso entrano i **linguaggi di IV generazione**, gli ultimi nati. Intendiamoci essi non nascono solo per l'end-user o solo per i Data Base, anzi hanno origine sostanzialmente nel tentativo di semplificare in generale la scrittura di qualunque tipo di applicazione. Addirittura si propongono di sostituirsi ai linguaggi evoluti, abbassando notevolmente i tempi di preparazione dei programmi: potremmo ad esempio vederli quasi come dei compilatori una cui frase quasi sostituisce, perché no?, da 3 a 7 statement COBOL.

Sono, comunque, parte del discorso perché è ovvio che se hanno queste caratteristiche sono assai adatti all'end-user e al lavoro sia su grande sistema che su personal computer.

Con ogni probabilità, l'ambiente di lavoro di domani vedrà un personal computer su ogni scrivania del personale non DP, il Data Base sarà distribuito ed ognuno potrà facilmente accedervi con un linguaggio come quelli della IV generazione.

Vedremo se il futuro ci darà ragione.

APPENDICE A

LA SCELTA DEI SUBSCHEMA

Una volta disegnato e descritto lo Schema, il Data Base administrator deve decidere quanti Subschema definire ed in quale modo essi dovranno essere strutturati. Tale decisione non è sempre agevole: si è spesso indecisi se convenga la definizione di un unico Subschema totale, o comunque di un numero limitato di grossi Subschema ai quali faranno riferimento più programmi o applicazioni, oppure se sia maggiormente conveniente un alto numero di Subschema di piccole dimensioni ad ognuno dei quali farà riferimento un numero più limitato di applicazioni.

Definire pochi Subschema di dimensioni notevoli presenta l'indubbio vantaggio di ottenere un dimensionamento totale delle applicazioni abbastanza limitato: non va dimenticato che nella memoria di ogni applicazione dovranno essere presenti le varie immagini di Subschema cui esse fanno riferimento. Si ha però il grave svantaggio di allargare la visione del Data Base con conseguente perdita di sicurezza; soprattutto accade che, nel caso di una modifica (e conseguente ritraduzione) di uno dei Subschema, si debbano ricompilare tutte le applicazioni (programmi, transazioni, ecc.) che vi fanno riferimento.

Di converso, lavorare con molti Subschema di piccole dimensioni presenta i vantaggi e gli svantaggi esattamente opposti: le dimensioni totali aumentano sensibilmente, ma diminuisce la necessità di numerose compilazioni nel caso di modifiche e, al contempo, l'indipendenza delle applicazioni (tra loro stesse) nonché la sicurezza (dovuta ad una visione più o meno limitata) divengono maggiori.

La situazione ottimale sembra essere quella intermedia, ma ovviamente la determinazione non è né semplice né immediata.

A nostro avviso il problema va posto in termini di opportuno raggruppamento (*clustering*): molte applicazioni hanno caratteristiche comuni per quanto riguarda l'uso che esse fanno dei vari tipi record; se si dispone delle informazioni relative a quali applicazioni usano certi tipi record, o viceversa quali record sono utilizzati nelle diverse applicazioni, si può giungere a determinare un metodo di raggruppamento o delle applicazioni, o dei record, o di entrambe le cose, che conduca alla definizione dei Subschema.

Prima di procedere, dunque, esaminiamo da vicino quali informazioni ci sono necessarie e come possiamo organizzarle.

In fase di analisi delle applicazioni bisogna che per ognuna venga annotata la presenza dei tipi record che interverranno nell'esecuzione. Ciò permetterà di ottenere una matrice come quella dell'esempio che segue: supponiamo siano 15 le applicazioni previste e che esse lavorino su un Data Base di soli 10 tipi record diversi, disponiamo le applicazioni nelle colonne ed i record nelle righe di una matrice 10 per 15, ponendo un contrassegno, ad esempio il valore 1, ad indicare che un certo record è utilizzato nell'applicazione corrispondente (vedi figura 1.1).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
101	–	1	1	–	1	–	–	–	–	1	–	–	–	–	–
102	–	1	1	–	1	–	1	–	–	1	–	–	–	–	–
103	–	1	1	–	1	–	–	–	–	1	–	–	–	–	1
104	–	1	1	–	1	–	–	–	–	1	–	1	1	1	–
105	1	1	1	–	1	–	–	1	–	1	–	1	1	1	1
106	1	–	–	1	1	1	1	1	–	–	1	–	–	–	–
107	1	–	–	1	–	1	1	1	1	–	1	–	–	–	–
108	1	–	–	1	–	1	1	1	1	–	1	–	–	–	–
109	1	–	–	1	–	1	1	1	1	–	1	–	–	–	–
110	–	–	–	–	–	–	–	–	–	–	–	1	1	1	1

Figura A.1. — La matrice che mostra l'utilizzo dei record nelle varie applicazioni. Queste ultime sono numerate da 1 a 15 e poste in colonna; i tipi record sono battezzati con i numeri da 101 a 110 e sono posti nelle righe.

Così il tipo record 101 è utilizzato nelle transazioni n° 2, 3, 5 e 10; il tipo record 102 nelle applicazioni n° 2, 3, 5, 7 e 10, e così via.

Diciamo subito che la raccolta di queste informazioni può risultare assai lunga se fatta a posteriori: è bene che tale matrice venga via via composta dall'analista nel momento in cui egli definisce l'applicazione, in tal modo essa scaturirà come risultato pressoché automatico dell'analisi stessa.

Ora si possono confrontare tra loro i vari vettori riga per verificare quanto essi stessi siano simili tra loro in funzione del loro utilizzo nelle diverse applicazioni. Oppure si può decidere di confrontare tra loro le diverse colonne per assimilare quelle applicazioni che utilizzano per lo più gli stessi record.

Anche una semplice occhiata può bastare, nel caso del nostro esempio, a farci concludere che esistono tre gruppi (vedremo poi a cosa ciò conduca), ma in casi maggiormente complessi non è detto che il mezzo visivo sia sufficiente e, soprattutto, mancando un preciso criterio oggettivo di decisione, non è detto che conduca a conclusioni corrette.

La scelta del punto di partenza è importante. Decidere se confrontare tra loro le colonne o le righe significa decidere di assimilare e quindi raggruppare le applica-

zioni secondo le loro caratteristiche funzionali e logiche, oppure, rispettivamente, di raggruppare i record.

In altre parole: nel primo caso si ottengono gruppi di applicazioni, si risale poi all'insieme dei record utilizzati in ogni gruppo: tali insiemi di record costituiranno le *strutture di base* dei vari Subschema. Nei vari gruppi comparirà una sola volta ogni applicazione, mentre uno stesso record potrà anche appartenere a più di un gruppo. Nel secondo caso, decidendo di assimilare i record, si ottengono gruppi in cui ogni record compare una sola volta, vale a dire si ottiene una cluaterizzazione in gruppi separati e i vari Subschema risulteranno disgiunti.

Per i nostri scopi il primo metodo sembra decisamente vantaggioso, specialmente quando si desideri, appunto, una visione del Data Base in strutture separate.

Riferendoci ancora all'esempio di figura A.1, esaminiamo cosa significhi notare, all'interno di quella matrice, la presenza di tre gruppi. Per fissare le idee, decidiamo di procedere confrontando tra loro le colonne (applicazioni). Si osserva abbastanza facilmente che le applicazioni n° 2, 3, 5 e 10 utilizzano più o meno gli stessi record, così le n° 1, 4, 6, 7, 8, 9, 11 ed infine le 12, 13, 14 e 15. I gruppi che giungiamo a formare sono quelli di figura A.2.

	Transazioni	**Record utilizzati**
Gruppo 1	**2,3,5,10**	**101,102,103,104,105,106**
Gruppo 2	**1,4,6,7,8,9,11**	**105,106,107,108,109**
Gruppo 3	**12,13,14,15**	**103,104,105,110**

Figura A.2. — Il raggruppamento dei record nel caso della figura A.1.

Un criterio oggettivo, tuttavia, per raggruppare le varie colonne, non può che essere affidato ad un coefficiente numerico. Una misura utile ed assai comoda nel valutare le somiglianze tra vettori ci è fornita dalla statistica: si tratta del coefficiente di correlazione "*r*". Non è questa la sede adatta per discutere del suo significato (per questo rimandiamo alla apposita letteratura), ci basti sapere che esso fornisce una valutazione numerica di quanto due vettori risultino simili in relazione a come sono distribuiti i valori numerici delle loro componenti. Se due vettori sono uguali si avrà il valore massimo (1), se i valori delle componenti sono distribuiti differentemente si avranno valori minori (fino a –1): se il valore è 0 si dice che i vettori non sono correlati, se il valore è –1 si dice che sono inversamente correlati.

Il calcolo è possibile attraverso una semplice formula e, specialmente se affidato all'elaboratore, assai veloce.

Si tenga presente che si potrebbe utilizzare qualunque altro legittimo metodo di valutazione, magari studiato ad hoc ed inventato al momento opportuno, per esegui-

re delle assimilazioni secondo i più svariati criteri. Ad esempio l'autore ha spesso utilizzato un coefficiente "*s*", che possiamo battezzare *di somiglianza*, che consiste nel conteggio delle *coincidenze di presenze*, ovvero nel contare quanti 1 occupano la stessa posizione nei due vettori, nel sottrarre poi il numero di *discordanze*, ovvero le situazioni di 0 e 1, o 1 e 0 nelle posizioni corrispondenti e quindi, tralasciando le *indifferenze*, 0 nelle stesse posizioni, nel normalizzare il valore ottenuto in modo da farlo risultare compreso tra −1 e 1. Questo secondo metodo presenta il vantaggio di fare assomigliare *meno* quei vettori che per l'alta presenza di zeri nella stessa posizione tendono a produrre coefficienti di correlazione abbastanza alti.

Non approfondiamo questo tipo di considerazioni, vediamo invece come si procede da questo punto in avanti, qualunque sia il coefficiente usato.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0.2	0.2	0.6	0.3	0.6	0.6	0.8	0.5	0.2	0.6	0.3	0.3	0.3	0.3
2		1	0.8	0.0	0.7	0	0.2	0.2	0.1	0.8	0	0.4	0.4	0.4	0.4
3			1	0	0.7	0	0.2	0.2	0.1	0.8	0	0.4	0.4	0.4	0.4
4				1	0.2	0.7	0.6	0.6	0.6	0	0.7	0.2	0.2	0.2	0.2
5					1	0.2	0.3	0.3	0	0.7	0.2	0.4	0.4	0.4	0.4
6						1	0.6	0.6	0.6	0	0.7	0.2	0.2	0.2	0.2
7							1	0.6	0.5	0.2	0.6	0.1	0.1	0.1	0.1
8								1	0.5	0.2	0.6	0.3	0.3	0.3	0.3
9									1	0.1	0.6	0.2	0.2	0.2	0.2
10										1	0	0.4	0.4	0.4	0.4
11											1	0.2	0.2	0.2	0.2
12												1	0.6	0.6	0.5
13													1	0.6	0.5
14														1	0.5
15															1

Figura A.3. — La matrice di somiglianza che si ottiene calcolando i coefficienti "*s*" per i vettori colonna della matrice di figura A.1.

I vettori colonna (ammettiamo, tanto per fissare le idee, di andare avanti così, ma come si è già notato, si potrebbe fare altrettanto agendo sulle righe) di una matrice come quella di figura A.1., vengono presi a due a due: il primo con tutti gli altri, poi il secondo con tutti gli altri e così via, calcolando per ogni coppia il coefficiente (di correlazione, o di somiglianza che sia). Si ottiene in questo modo un insieme di valori che possiamo disporre in una matrice quadrata simmetrica in cui l'elemento "*e(m,n)*" della riga "*m*" e colonna "*n*" è il coefficiente calcolato per la colonna "*m*" e la colonna "*n*" della matrice di partenza. Chiamiamo *di correlazione* o *di somiglianza* tale matrice e notiamo che la diagonale principale deve essere composta dai valori massimi (automaticamente 1 se il coefficiente usato era quello di correlazione, altrimenti tale valore vi dovrà essere forzato) poiché rappresenta la somiglianza di un vettore con se stesso.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	–	–	*	–	*	*	*	*	–	*	–	–	–	–
2		1	*	–	*	–	–	–	–	*	–	*	*	*	*
3			1	–	*	–	–	–	–	*	–	*	*	*	*
4				1	–	*	*	*	*	–	*	–	–	–	–
5					1	–	–	–	–	*	–	*	*	*	*
6						1	*	*	*	–	*	–	–	–	–
7							1	*	*	–	*	–	–	–	–
8								1	*	–	–	–	–	–	–
9									1	–	*	–	–	–	–
10										1	–	*	*	*	*
11											1	–	–	–	–
12												1	*	*	*
13													1	*	*
14														1	*
15															1

Figura A.4 — La matrice di figura A.3. contrassegnata per un valore di soglia 0.30.

Esaminiamo, sempre per il caso da cui siamo partiti (figura A.1.), la matrice di somiglianza: in essa ci è permesso, ovviamente, di tenere conto solo della metà superiore (vedi figura A.3.). Fissiamo un valore, che diciamo *di soglia* e che, almeno per il momento, possiamo ritenere arbitrario, ed eseguiamo una scansione della matrice contrassegnando tutti gli elementi maggiori di quel valore.

Così, ad esempio, fissato il valore di soglia 0.30, gli elementi con valore maggiore sono quelli che nella figura A.4. sono segnati come asterisco, degli altri non teniamo addirittura conto.

Tutti gli asterischi vanno pensati come un legame: cioè, se ad esempio consideriamo la quarta riga di figura A.4., vediamo che essi sono presenti nelle colonne 6, 7, 8, 9 e 11.

Da ciò concludiamo che i vettori delle applicazioni n° 4, 6, 7, 8, 9 e 11 vanno fatti appartenere allo stesso gruppo (ma magari procedendo nella scansione si scopriranno altre relazioni e quindi altri elementi da inserire nello stesso gruppo).

Per visualizzare meglio la situazione usiamo la comoda tecnica dei grafi: simbolizziamo con un punto numerato ogni vettore e con una linea il legame, cioè l'esistenza di un coefficiente, calcolato su di essi, maggiore del valore di soglia.

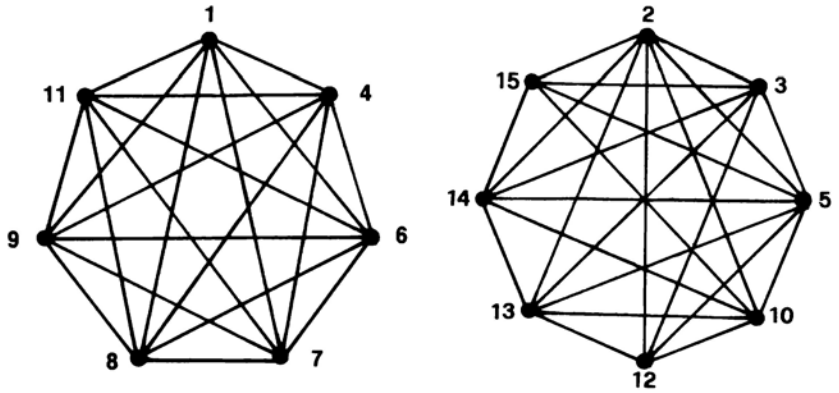


Figura A.5 — Il grafo per il valore di soglia 0.30.

Come si vede facilmente i legami si sono stabiliti in modo tale da determinare due gruppi ben distinti. Ma non basta: è lecito attendersi che facendo le stesse cose con un valore di soglia superiore, alcuni di quei legami si spezzino conducendo alla determinazione di un maggior numero di gruppi. Vediamo come si modifica la matrice contrassegnata qualora venga preso il valore di soglia 0.45 (vedi figura A.6.).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	–	–	*	–	*	*	*	*	–	*	–	–	–	–
2		1	*	–	*	–	–	–	–	*	–	–	–	–	–
3			1	–	*	–	–	–	–	*	–	–	–	–	–
4				1	–	*	*	*	*	–	*	–	–	–	–
5					1	–	–	–	–	*	–	–	–	–	–
6						1	*	*	*	–	*	–	–	–	–
7							1	*	*	–	*	–	–	–	–
8								1	*	–	*	–	–	–	–
9									1	–	*	–	–	–	–
10										1	–	–	–	–	–
11											1	–	–	–	–
12												1	*	*	*
13													1	*	*
14														1	*
15															1

Figura A.6. — La matrice di figura A.3. contrassegnata per un valore di soglia 0.45.

Confrontando con figura A.4. si vede come siano andati perduti diversi legami: non sono più presenti gli asterischi in (2,12), (2,13), (2,14), (2,15), (3,12) e così via. Ciò non può che corrispondere ad un aumento del numero di gruppi che conterranno un numero minore di elementi. Visualizzando ancora con i grafi, si nota come il secondo dei gruppi di figura A.5. si sia suddiviso in due gruppi distinti (vedi figura A.7.).

Ci si renderà conto, ora, che proseguendo secondo questo criterio con valori sempre maggiori del coefficiente di soglia si giunge ad una separazione sempre maggiore, cioè ad un numero di gruppi sempre più grande, fino ad arrivare alla situazione di 15 gruppi, ognuno composto di un solo elemento. A quest'ultimo punto si giungerebbe obbligatoriamente solo volendo considerare anche valori di soglia maggiori di 1, altrimenti almeno gli elementi tra loro uguali (composti di tutti 1 e con coefficiente = 1) devono permanere comunque sempre nello stesso gruppo.

Già ora ci si accorge che si possono ottenere quanti raggruppamenti si desiderino e che è sufficientemente agevole, una volta stabilito un gruppo, risalire a quali record siano interessati nonché alle dimensioni del gruppo stesso. Tuttavia la cosa più

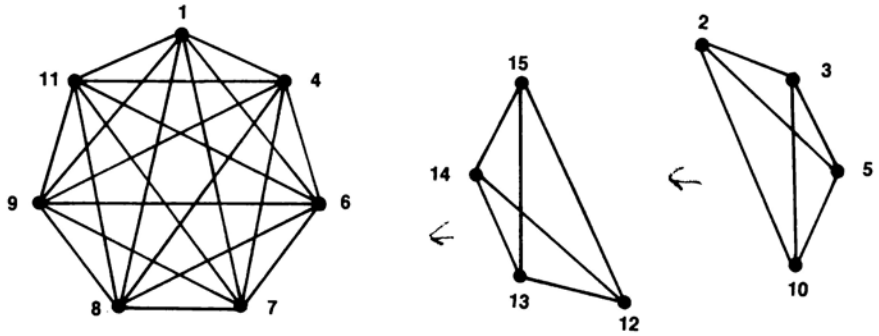


Figura A.7. — Il grafo per il valore di soglia 0.45.

importante è notare, a questo punto, che sarà la scelta di un valore di soglia del coefficiente a determinare le conclusioni e che quindi saranno necessarie alcune considerazioni che permettano di evitare quel carattere di arbitrarietà della scelta sulla cui base si è finora proceduto.

Supponiamo di eseguire la scansione della matrice *di somiglianza* ripetutamente per valori di soglia che aumentano gradatamente e di compilare una tabella delle corrispondenze tra il valore ed il conseguente numero di gruppi: essa prenderà nome di *tabella di raggruppamento*. La lunghezza di tale tabella dipenderà sia dal valore di soglia iniziale che da quello finale (normalmente 0 e 1), ma soprattutto dalla grandezza dell'incremento che si assegna ciclicamente a tale valore.

Nel nostro esempio supponiamo di partire dal valore 0.0 e, incrementando di 0.05, di giungere fino ad 1. Si ottiene una tabella di raggruppamento come quella di figura A.8.

Valore del coefficiente	Numero dei gruppi	Valore del coefficiente	Numero dei gruppi
0.05	1	0.55	4
0.10	1	0.60	4
0.15	1	0.65	5
0.20	1	0.70	9
0.25	1	0.75	12
0.30	2	0.80	15
0.35	2	0.85	15
0.40	3	0.90	15
0.45	3	0.95	15
0.50	3	1.00	15

Figura A.8. — La tabella di raggruppamento relativa al nostro esempio.

Come si nota, inizialmente tutti gli elementi appartengono ad un unico gruppo, poi, via via, questo si suddivide in 2, 3, ... gruppi; nel passaggio da 0.70 a 0.75 si ha una brusca impennata che porta a 12 e quindi a 15 gruppi.

Ovviamente più fine è la scansione (cioè, minore l'incremento) meglio definita risulta la situazione.

Già in un caso come quello dell'esempio, tuttavia, ci si può rendere conto di un fatto: l'aumento del numero dei gruppi procede prima abbastanza lentamente, poi molto velocemente. Se riportiamo la tabella in un grafico in cui si pongono in ascisse i coefficienti ed in ordinata il numero di gruppi, si nota che prima della brusca salita si ha sempre un *pianerottolo* lungo (il più lungo prima della salita) quanto basta a farci ritenere che quella sia la situazione di raggruppamento ottimale, la meglio rispondente alla struttura del nostro problema.

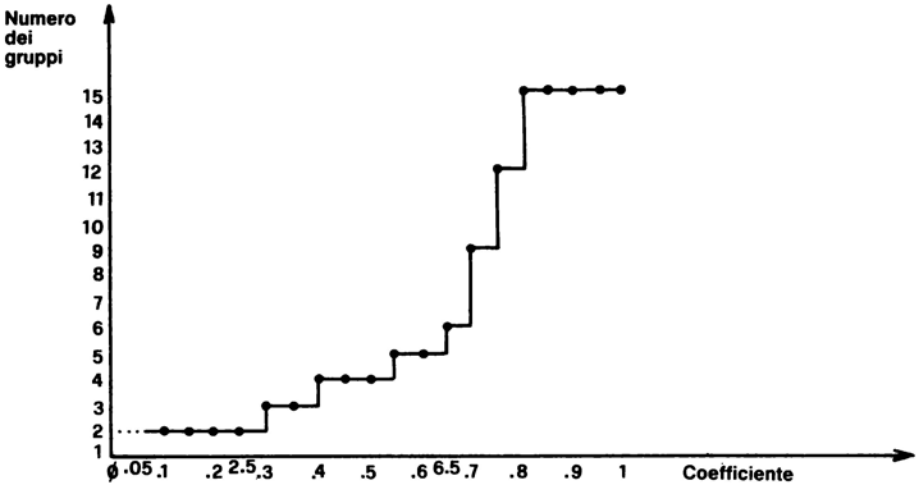


Figura A.9. — Il grafico che rende conto del numero dei gruppi in funzione del valore del coefficiente.

In effetti nel nostro caso, come si vede in figura A.9, escludendo la permanenza ad 1 ed a 15 gruppi, la permanenza maggiore si ha sui 3 gruppi. Più in generale si è verificato, applicando il metodo a vari casi e con scansioni sufficientemente piccole, che si ottiene praticamente sempre una situazione come quella di figura A.10.

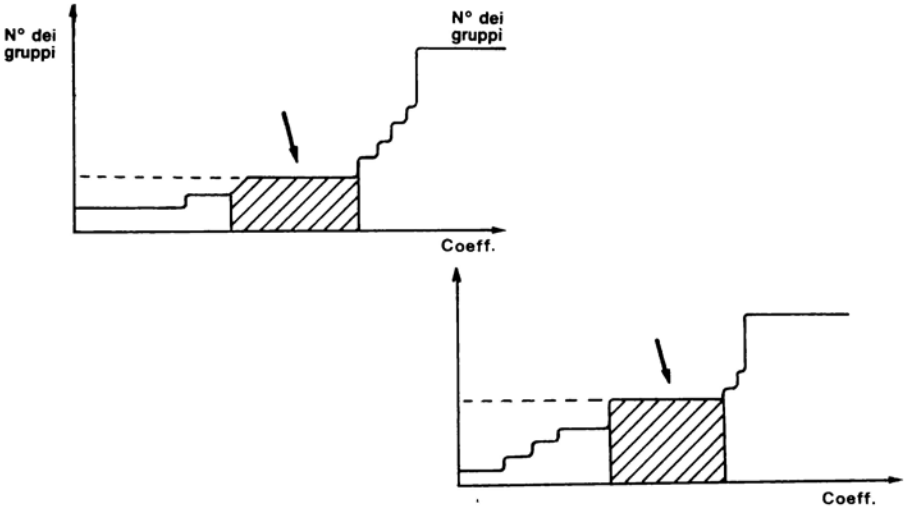


Figura A.10. — Si è verificato, sia pure sperimentalmente, che le distribuzioni "*N° dei gruppi - coefficiente*" hanno sempre un andamento come quello descritto, in cui è facile individuare una "*permanenza significativa*" (qui indicata dalla freccia).

Le permanenze indicate dalle frecce, che diremo *permanenze significative*, mostrano il numero di gruppi ottimale. Esso potrà essere, nei vari casi, più o meno evidente, ma sarà comunque facilmente individuabile.

Il metodo procederà per i passaggi analizzati: verrà deciso il valore iniziale e finale del coefficiente di soglia, nonchè il passo d'incremento per una scansione più o meno fine. Il risultato di questi passaggi di scansione delle matrici consisterà nella produzione delle *tabelle di raggruppamento*; a questo punto verrà richiesta la scelta del coefficiente di soglia desiderato, ovvero di uno di quelli corrispondenti alla *permanenza significativa*. Sulla base di tale scelta verrà prodotta l'evidenza dei vari gruppi (i record che intervengono in ognuno). La scelta del coefficiente potrà essere ripetuta: ciò potrà risultare utile in quei casi in cui sembra vi sia più di una permanenza significativa, o paia ve ne sia più d'una.

Può essere interessante notare che ripetendo il procedimento sia sulla matrice di somiglianza sia su quella di correlazione, si ottengono sempre delle situazioni congruenti, in cui, come in figura A.11, i due diagrammi, relativi alle due diverse analisi, si sovrappongono proprio sulla permanenza significativa, agevolandone il riconoscimento.

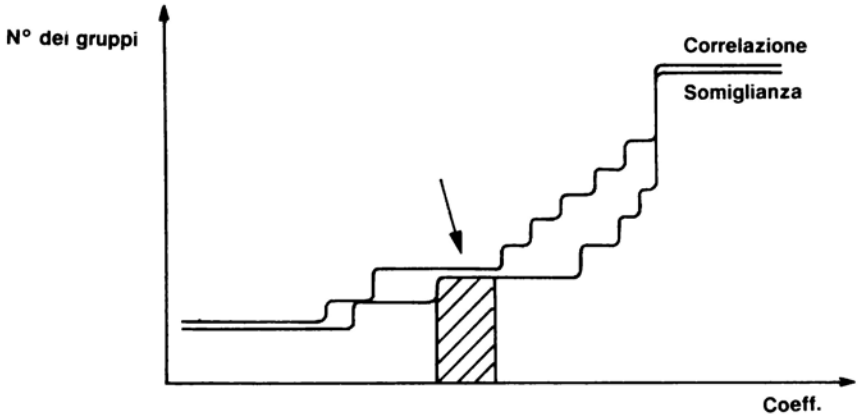


Figura A.11. — Procedendo col coefficiente di correlazione o di somiglianza i risultati sono gli stessi almeno per quanto riguarda il riconoscimento dei coefficienti corrispondenti alla "*permanenza significativa*".

Potrebbe sembrare che la scelta del coefficiente di soglia non abbia perso, in relazione alla determinazione della permanenza significativa, il suo carattere di arbitrarietà: quanto meno, ci si potrebbe obbiettare, essa è soggetta ad una decisione umana, forse non del tutto oggettiva. Va invece tenuto presente che la permanenza significativa è effettivamente una caratteristica statistica dei raggruppamenti, presente anche nei casi più complessi: nei rarissimi casi in cui la permanenza significativa non compare (ogni permanenza è più o meno della stessa lunghezza) è legittimo concludere che le caratteristiche della matrice di partenza sono tali da non poter condurre a raggruppamenti basati su analogie logico-funzionali delle varie applicazioni. Ciò è come dire che ogni applicazione è un caso a sè, che non ha nulla di simile alle altre per quanto riguarda l'uso che essa fa delle informazioni di Data Base: ciò, appunto, non è praticamente mai vero.

Anche in questi eventuali rarissimi casi, tuttavia, il metodo potrebbe risultare assai utile: sarebbero evidenti i possibili raggruppamenti, sarebbero più facilmente riconoscibili ed analizzabili i possibili Subschema e ne conosceremmo immediatamente le dimensioni: la scelta del coefficiente di soglia dovrà essere affidata al buon senso piuttosto che ad un criterio specifico, ma sarà enormemente facilitata.

Va ancora fatta un'osservazione generale: non è detto che i Subschema ottenuti dai raggruppamenti vadano ritenuti definitivi. Solo se l'analista avesse tenuto presenti tutte le caratteristiche funzionali delle applicazioni nel compilare la matrice iniziale, nonchè della struttura del Data Base (ovvero del *disegno*), avremmo la sicurezza della completa correttezza dei gruppi risultanti.

Non è detto che sia solo accaduto di aver dimenticato qualcosa, più spesso può essere successo che si siano raggruppate le informazioni per quello che sono, senza tener conto, ad esempio, che per accedere ad un record si debba passare per un set cui appartengono altri record. Dopo una verifica, ciò ci costringerà a modificare la matrice di partenza inserendo anche gli elementi mancanti. Anzi, potrebbe essere buona norma inserire nella matrice iniziale qualunque tipo di elemento: set, tipo di accesso, ecc., fin dall'inizio. Ciò permetterebbe sicuramente un miglior utilizzo del metodo. Tuttavia ... al lettore ed utilizzatore il mediare, da una parte, tra semplicità ed immediatezza di visione e, dall'altra, completezza e correttezza.

Un'ultima annotazione: si sarà sicuramente osservato come il metodo sia automatizzabile, lasciando all'elaboratore tutte le faticose fasi di calcolo e riconoscimento dei gruppi (*). Proprio così facendo esso dimostra maggiormente la propria utilità.

(*) Così ha proceduto l'autore. Prove e verifiche sono state fatte dopo una automatizzazione del metodo in alcuni programmi FORTRAN di facile esecuzione ed utilizzo.

APPENDICE B

DATA BASE E DOCUMENTAZIONE: IL DIZIONARIO DEI DATI

B.1 — Dizionari di dati

Già si è avuto occasione di notare che il Data Base è uno strumento utile in molte occasioni: ci permette di archiviare le informazioni da elaborare, di memorizzare tutto quanto può servire alla nostra organizzazione, di risolvere i più svariati problemi. Ora, l'osservazione che sorge spontanea è che esso può utilmente soccorrerci anche in uno dei problemi più grossi che l'elaborazione automatica delle informazioni attualmente propone quello della *documentazione*.

Sicuramente vale la pena esaminare perchè si parli di problema, quale sia il senso della parola *documentazione* e come un Data Base possa entrare nel discorso.

Molti potrebbero pensare che il costo maggiore nella organizzazione di un centro DP sia quello dovuto allo sviluppo di nuove applicazioni. Non è così e l'idea che una buona documentazione possa rendere il lavoro assai più economico è da ritenersi ampiamente giustificata. Cerchiamo di analizzare brevemente perchè.

Quando la meccanizzazione era agli inizi, l'elaboratore era una macchina assai complessa: difficile da usare richiedeva un personale di tipo assai particolare, capace di adattare alla macchina le richieste umane. Al contempo le esigenze applicative erano relativamente limitate. In questa situazione il costo di un'applicazione era assorbito per la maggior parte del suo sviluppo e dalla sua successiva gestione. Prendendo uguale a 100 il costo globale di un'applicazione EDP, una valutazione fatta attorno alla metà degli anni 60 poteva vedere i costi di manutenzione aggirarsi attorno al 35% (vedi figura .. a). Solo circa quindici anni più tardi la situazione risultava radicalmente cambiata: i costi di manutenzione potevano allora venir valutati anche attorno al 70% (vedi ancora figura B.1.a).

Indubbiamente si potrebbe discutere molto su queste percentuali: la situazione poteva variare di parecchio caso per caso ed un centro già ben organizzato sarebbe stato sicuramente in grado di rispondere in maniera più adeguata ed economica alle esigenze di manutenzione. Sta di fatto, tuttavia, che era concretamente apprezzabile una tendenza di fondo: col passare del tempo la manutenzione finiva via via con l'assorbire percentualmente i costi maggiori.

A qualche anno ancora di distanza, l'esperienza ha dimostrato che non si tratta

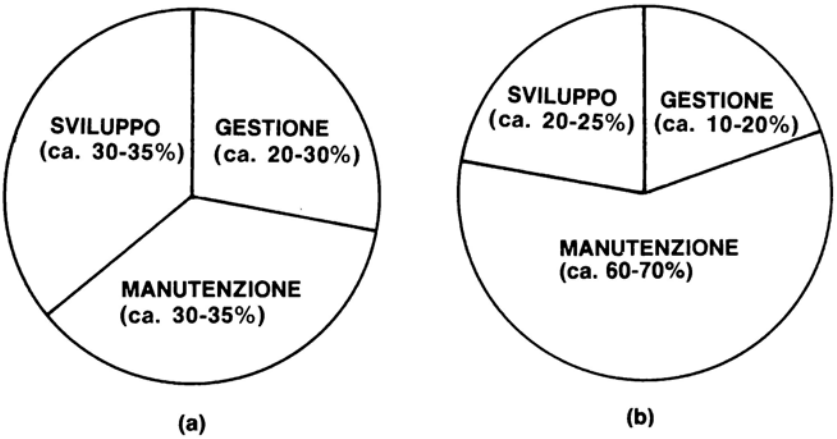


Figura B.1.a — VALUTAZIONE DELLA RIPARTIZIONE DEI COSTI DI UNA APPLICAZIONE. Nell'attività di SVILUPPO vanno incluse: definizione delle richieste, disegno, progetto dell'applicazione e sua realizzazione. L'attività di GESTIONE è costituita quasi unicamente dall'utilizzo periodico. Le fasi di MANUTENZIONE riguardano i miglioramenti, le estensioni a nuovi campi di applicabilità, in generale tutte le modifiche di cui si presenti la necessità. La situazione (a) corrisponde ad una valutazione come poteva venir effettuata più o meno agli inizi dell'era informatica, quando la realizzazione di un'applicazione e la sua gestione richiedevano un impegno notevole. A pochi anni di distanza — situazione (b) — i costi di manutenzione potevano essere ritenuti preponderanti. Oggi si cerca di far diminuire tali costi con metodi di lavoro più opportuni, con metodologie ben organizzate e standardizzate (come, ad esempio, con la programmazione strutturata) e soprattutto dando l'opportuno peso agli aspetti documentativi.

solamente di organizzare bene il lavoro, ma che il crescere della complessità delle applicazioni ed il loro evolversi finiscono comunque per causare sforzi assai più sensibili nelle fasi di modifica o, più in generale, di mantenimento che non nella fase della loro stessa progettazione e realizzazione. Bisogna anzi riconoscere che tutt'oggi questo è uno dei problemi più importanti e che alla sua soluzione è devoluto l'impegno maggiore.

Vale sicuramente la pena analizzare un po' più in dettaglio almeno le principali tra le cause di questa situazione. È innegabile che: il normale *turn-over* aziendale provoca in questo campo scompensi notevoli, che le variazioni dell'ambiente in cui

un'applicazione vive (nuove leggi, ristrutturazioni aziendali, nuovi ordinamenti) non possono aver termine ed infine che la normale evoluzione richiede sempre nuovi sviluppi applicativi che devono tener conto di quanto già esiste.

Per quanto riguarda il primo punto va precisato che i danni notevoli causati dal normale ricambio di personale sono intimamente collegati al modo di lavorare ed al tipo di organizzazione; tuttavia, in ogni caso, ciò che un dipendente ha prodotto è quasi sempre assai difficilmente gestibile e modificabile da altri. Quante volte è accaduto ed accade che una persona, trovandosi a gestire procedure preparate da altri, si sia trovata in difficoltà eccessive? Sicuramente troppo spesso. L'osservazione che una buona documentazione avrebbe risolto o risolverebbe in buona misura il problema, è addirittura banale.

Per quanto poi riguarda la questione delle modifiche, che sono sempre necessarie ed in una certa quantità, va tenuto presente che è la normale evoluzione dell'azienda (ma, oserei dire, il normale dispiegarsi della pratica quotidiana) a produrre richieste sempre nuove e che un'applicazione non è mai una cosa fissa, stabile, definita una volta per tutte: essa vive, cambia, si evolve così come si evolvono le esigenze aziendali, così come cambiano la realtà ed i problemi cui essa fa fronte. L'alta richiesta di modifiche che normalmente si incontra nelle organizzazioni contemporanee è destinata a crescere e rappresenta un fattore importante da non trascurare. Ancora la risposta al problema è banale: procedure ben documentate, precedute da analisi altrettanto ben documentate, sono facilmente gestibili e modificabili, procedure a sè stanti, impiantate come macchine che dovrebbero funzionare da sole, senza ombra di spiegazione su come sono nate e su come lavorano, sono cose da buttare come inutili, o sono misteriosi marchingegni destinati comunque ad una morte prematura.

Ma che dire delle procedure nuove, in via di sviluppo, o addirittura dei nuovi progetti? Se si parte da zero il problema può essere ridotto a quello di fare delle corrette previsioni su quello che diverrà la nostra organizzazione, ma se esistono già degli archivi, un Data Base, altre procedure più o meno complesse, non tener conto di ciò che già esiste è sicuramente un errore sia perchè si rischia di rifare cose già fatte, sia perchè si rischia di duplicare delle informazioni e di finire col rendere ridondante tutto il nostro sistema informativo. Come evitarlo se ciò che già esiste non è ben documentato? Come si vede, in tutti i casi la risposta al problema è una buona documentazione.

In effetti si può pensare che, in termini di costi, il trend evolutivo di un'applicazione subisca un andamento come quello dei grafici di figura B.1.b. Dal loro confronto risulta visivamente evidente come la presenza di certe informazioni descrittive e di uno standard di lavoro che prevede una buona documentazione in ogni fase permetta, nel tempo, una maggiore economia. Qualora le applicazioni siano prive di documentazione ogni fase di modifica produce un aumento dei costi sensibilmente elevato ed una buona documentazione costa di più soltanto in fase di realizzazione iniziale.

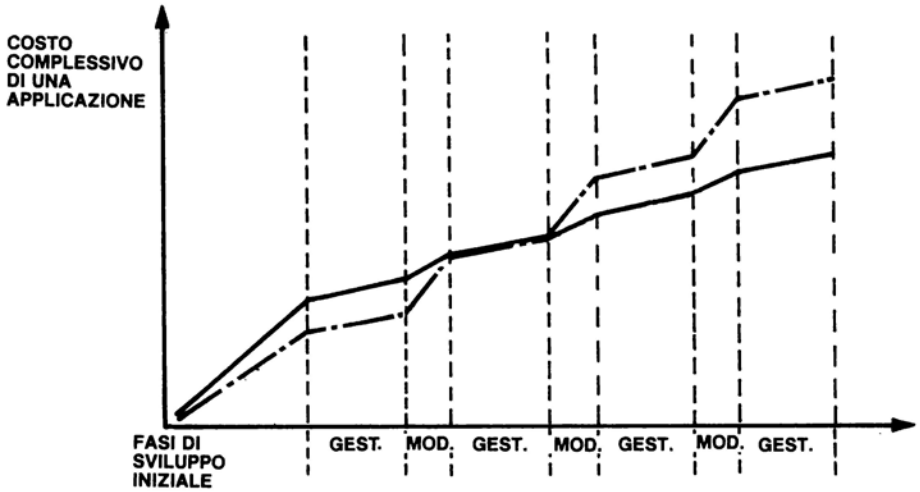


Figura B.1.b — ANDAMENTO DEI COSTI DI UNA APPLICAZIONE.
Si può ritenere che il costo complessivo di un'applicazione abbia nel tempo un andamento come quello indicato. Un'applicazione tenderà, a regime, a mantenere costante il proprio costo, ma nel periodo iniziale si possono avere aumenti sensibili. In particolare sono le fasi di modifica a provocare gli incrementi maggiori. Una applicazione che goda di una buona documentazione (grafico inizialmente superiore) costa di più solo nella fase di sviluppo, proprio per la preparazione della documentazione stessa. Una che ne sia priva (grafico inizialmente inferiore) può condurre a situazioni economicamente svantaggiose. Assumendo che i costi di gestione provochino aumenti scarsamente apprezzabili, ma comunque identici nei due casi, le fasi di modifica risultano per l'una assai meno dispendiose che per l'altra e ciò al punto che, col passare del tempo, l'applicazione ben documentata finisce per risultare assai più economica.

Una situazione di lavoro ottimale può essere ottenuta in vario modo: ad esempio stabilendo degli standard aziendali che analisti e programmatori dovranno poi rispettare abbastanza rigidamente aggiungendo tutte le descrizioni di ciò che viene fatto in maniera completa e coerente, oppure utilizzando delle opportune metodologie alle quali, poi, ogni fase applicativa dovrà adeguarsi. Proprio riguardo a quest'ultimo punto vi sarebbe parecchio da dire: si pensi solamente che è anche per rispondere a questo tipo di esigenze che sono nati vari strumenti e metodi standard di analisi e soprattutto le varie metodologie di programmazione strutturata.

Comunque, per tornare alle informazioni di tipo descrittivo, va notato che la documentazione è stata praticamente sempre affidata a supporti di tipo cartaceo. La cosa è ovvia per quanto riguarda il dossier di analisi, le descrizioni delle richieste, il disegno generale dell'applicazione; lo è altrettanto per quanto riguarda le descrizioni delle parti procedurali (ed i commenti stessi che vi vengono inseriti), ma lo è meno per le strutture informative, ovvero per i dati. Per tutto ciò che riguarda l'input-output e gli archivi viene normalmente conservata una descrizione su moduli di vario tipo più o meno ben catalogati. Ma il supporto cartaceo soffre di un difetto costituzionale piuttosto grave: tende a diventare troppo voluminoso, ingombrante e spesso il reperimento delle informazioni diviene difficoltoso; esso è facilmente deperibile, aggiornarlo può diventare addirittura arduo ed infine non è facilmente proteggibile da consultazioni o interventi indesiderati.

Ovviamente tutto ciò non significa che documenti di tal genere siano del tutto svantaggiosi: almeno una parte di essi non smetterà mai di essere utile se non necessaria. Tuttavia da molte parti si è ormai sentita la necessità di affidare all'elaboratore stesso la conservazione e la gestione della maggior parte delle informazioni di tipo documentativo ottenendo così il vantaggio di minimizzarne l'ingombro ed al contempo di automatizzarne il trattamento.

Diversi utenti si sono costruiti, ormai da tempo, il proprio archivio automatizzato dei *dati sui dati*, ovvero un archivio delle informazioni relative a tutta la propria organizzazione, nonchè tutte le procedure di intervento utili.

Spesso si è trattato di veri e propri Data Base nati con il preciso scopo di accogliere una mole di dati notevole e di permetterne un trattamento agevole consentendo le necessarie protezioni. Archivi di questo tipo sono normalmente chiamati *dizionari dei dati*, o più spesso con il corrispondente anglosassone *data dictionary*. Molte case di consulenza hanno già provveduto alla loro realizzazione ed alcune case costruttrici, quelle che hanno fatto altrettanto, danno un particolare rilievo alla loro produzione di software applicativo proprio grazie a questo strumento.

Analizzeremo un esempio specifico di dizionario nel prossimo paragrafo (*). Prima, tuttavia, vedremo di tentare una più precisa definizione delle caratteristiche generali che un data dictionary deve avere, di dare una descrizione delle sue funzionalità e di considerare i problemi che è in grado di risolvere.

Un data dictionary può essere definito come un archivio in cui siano memorizzati tutti gli elementi capaci di interessare in qualche modo il sistema informativo, nonchè le relazioni che tra essi intercorrono. In particolare in esso saranno riassunte, come già si è accennato, tutte le informazioni relative ai dati, al loro formato ed al loro utilizzo nei diversi contesti applicativi. Le diverse categorie di elementi vengono chiamate normalmente *tipi entità*, ogni singolo elemento dovrà avere un suo nome univocamente definito e si parlerà di *entità singola*. Ognuna di queste potrà godere di una larga serie di *attributi*: sono questi che la caratterizzano e ne docu-

(*) Discuteremo di quello recentemente messo a punto dalla Honeywell.

mentano in vario modo il significato. Le relazioni che da un punto di vista logico od organizzativo intercorrono tra i vari elementi saranno stabilite *connettendo* tra loro le entità singole.

Indubbiamente una struttura che rispetti questa definizione e queste caratteristiche può essere realizzata in molti modi anche assai diversi tra loro, ciò che conta è che poi sia utilizzabile in modo agevole. Si osservi la figura B.1.c: in essa si è cercato di suggerire uno schema del nucleo di tale struttura. Naturalmente possono esistere schematizzazioni completamente diverse che conducono a soluzioni altrettanto efficaci, dipenderà dalla fantasia del progettista. Tuttavia quella presentata offre sicuramente il vantaggio di una grande semplicità e di una notevole facilità di realizzazione. Il modo in cui ottenere tale realizzazione è intimamente collegato al tipo di software di cui si dispone, ma è abbastanza immediato accorgersi che le attuali tecniche che fanno ricorso ai Data Base (ed in particolare a quelli integrati) possono venirci in aiuto: si tratta di costruire un archivio che soddisfi certe esigenze logiche, che permetta la gestione di volumi notevoli e la massima centralizzazione.

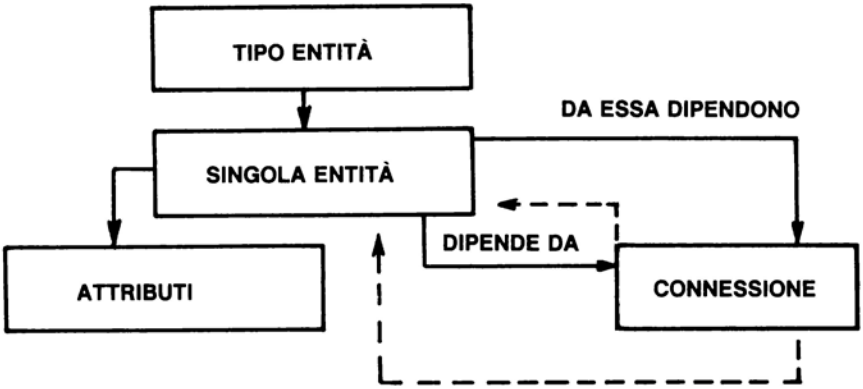
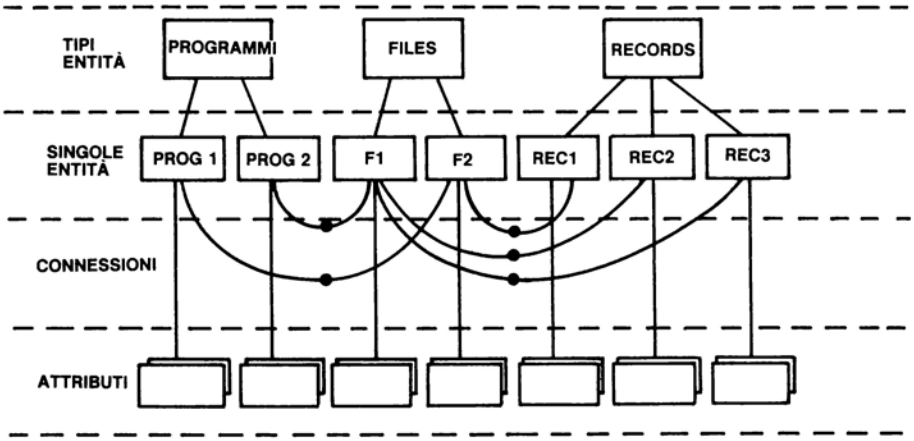


Figura B.1.c — IL NUCLEO DI UN DIZIONARIO

Ad ogni tipo entità corrisponde un certo numero di entità singole ognuna delle quali ha una serie di attributi. Ogni singola entità può essere collegata ad altre: una entità può dipendere da un'altra, o, che è lo stesso, quest'ultima le è gerarchicamente superiore nella logica dell'organizzazione o dello specifico ambiente applicativo. Questo schema non è che uno dei possibili ed è abbastanza parziale poichè riguarda solamente la parte centrale del problema: nulla, ad esempio, impedisce che vengano stabilite connessioni più estese od aggiunti elementi di vario genere a seconda delle particolari esigenze. Tuttavia si noti come già esso permetta di risolvere il problema di documentare quali programmi utilizzino certi files e di quali records questi ultimi siano composti; ad esempio:



in questo caso le entità previste come tipo sono state: programmi, files e records; gli attributi (aggiunti solo indicativamente) hanno carattere descrittivo.

Per quanto riguarda una sorta di dimostrazione della necessità di disporre di una data dictionary, è bene ricordare che quando si abbia il sentore di una qualche difficoltà nel mantenere il controllo delle risorse informative, un intervento di tipo tradizionale (quali una riorganizzazione della documentazione esistente o, più in generale, una ristrutturazione delle risorse stesse) può risultare già tardivo. Forse il primo momento in cui ci si accorge di tale difficoltà è quello in cui si nota una certa lentezza nella determinazione delle informazioni di cui si deve disporre a fronte di certe modifiche: è l'ultranoto problema del dato che cambia e che non si sa più in quali applicazioni sia utilizzato o, alla fine, in quali programmi compaia. Ci si potrebbe chiedere se si sarebbe potuto ovviare a tutto ciò: rispondere che il rimedio consiste in una buona documentazione, questa volta non basta. C'è da tenere presente che per essere efficace questa documentazione deve risultare il più possibile centralizzata e di agevole accesso. Ecco dunque che un buon dizionario, che di tali caratteristiche non può mancare (anzi esse ne costituiscono gli aspetti peculiari), risponde al problema in maniera soddisfacente.

Tutti i discorsi di carattere introduttivo sui problemi risolti dalla documentazione volevano, in definitiva, condurre a questa conclusione: *un dizionario ci soccorre in maniera ineguagliabile sia nel realizzarla che nel gestirla.*

Prima di concludere, tuttavia, esistono altre due argomentazioni, tra loro collegate, che dimostrano quanto un dizionario risolva problemi reali: si tratta della standardizzazione e delle ridondanze. Di esse vale la pena dare almeno un cenno.

Sicuramente i problemi che conducono ad un difficile rispetto degli standard sono di ordine strutturale ed intimamente legati alla situazione del personale. Altrettanto sicuramente, però, un archivio centralizzato al quale fare abitualmente ricorso almeno per quanto riguarda le definizioni, i nomi, le strutture dei dati comuni, facilita e rende possibile quel rispetto. Quanto poi alle ridondanze si potrebbero spendere fiumi di parole affrontando questioni anche complesse e portando ad esempio situazioni reali di sistemi informativi divenuti inefficienti per loro causa. Basterà tuttavia notare che quasi sempre una ridondanza (uno spreco di spazio oppure la ripetizione di un nome) ne provoca altre e poi altre ancora e così via in un crescendo inarrestabile. Inutile aggiungere che proprio la centralizzazione delle risorse informative in un dizionario ci evita tutto ciò.

A questo punto, per vedere più concretamente come un dizionario intervenga nella soluzione di tutti i problemi succitati, è bene analizzarne un esempio. Lo faremo esaminandone brevemente le fasi di planning, la realizzazione e le particolari funzionalità.

B.2 Realizzazione e funzionalità di un dizionario dei dati

Indipendentemente dal tipo di software di cui si dispone, la fase di pianificazione di un dizionario deve rispondere a criteri ben precisi di organizzazione e standardizzazione. È possibile elencare una serie di passi che in questa fase iniziale è bene comunque compiere e che permettono di procedere in modo ordinato e coerente. Quanto segue al riguardo rappresenta una serie di consigli e di atteggiamenti da adottare, non si pretende né di stabilire uno standard di progettazione, né di proporre soluzioni miracolose ai problemi d'analisi. Si tenga presente, tuttavia, che quando ci si occupa di problemi di questo tipo, il lavoro procede secondo criteri specifici, ha caratteristiche peculiari ed abbastanza diverse di quelle relative all'analisi di una normale applicazione.

I punti focali della pianificazione (vedi figura B.2.a) riguardano prima di tutto la traduzione delle esigenze in funzioni eseguibili: le varie richieste vanno attentamente analizzate al fine di giungere ad una definizione quanto più precisa possibile dei modi e dei metodi per ottenerle. Questo primo compito va visto composto di due fasi distinte: la prima deve consistere in una *analisi delle richieste di informazione da parte della organizzazione* e la seconda in una vera e propria *definizione dei modelli funzionali.*

Esaminiamo allora la prima e procediamo in modo abbastanza dettagliato poiché essa riveste, è immediato comprenderlo, un'importanza del tutto particolare e condiziona tutto ciò che segue.

Utenti del *data-dictionary* possono essere tutte le componenti aziendali. Non si deve pensare che ciò che è raccolto nel nostro dizionario debba riguardare solo settori particolari: ogni ufficio (anche non DP), ogni gruppo di lavoro può ricorrere vantaggiosamente alle informazioni documentative che vi abbiamo archiviato. Quindi un

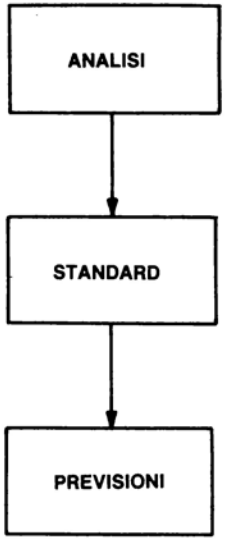


Figura B.2.a — I punti ai quali è bene badare durante la pianificazione del dizionario riguardano, essenzialmente, la traduzione delle necessità organizzative in funzioni eseguibili, la definizione degli standards ed una previsione delle possibilità di crescita.

primo aspetto: la nostra indagine sulle richieste deve risultare quanto più possibile vasta e completa. Un secondo aspetto: la nostra indagine deve risultare estremamente accurata. Ogni richiesta andrà identificata nei suoi dettagli e documentata, ciò condurrà alle specifiche sia strutturali che quantitative utili al disegno del dizionario, cioè condurrà alla definizione di quali entità, quali attributi, quali relazioni andranno stabilite e su quali dimensioni si dovrà lavorare. Contemporaneamente sarà bene prevedere subito altre due cose importanti: la protezione da accessi indesiderati alle informazioni e la frequenza con cui a quelle informazioni si dovrà far ricorso. Ciò soprattutto per non correre il rischio di dover manipolare in un secondo tempo quanto già prodotto.

Quanto poi alla seconda fase, quella di definizione dei modelli funzionali, va detto questo: previa una perfetta conoscenza di ciò che il software del nostro dizionario permette, risulta utile una suddivisione delle richieste in gruppi funzionali distinti. In ognuno di essi sarà riconoscibile l'*attività* da svolgere per il raggiungimento dell'obiettivo, ovvero sarà possibile definire la *unità funzionale eseguibile*. In questa fase è consigliabile procedere anche alla definizione dei supporti più opportuni per le singole informazioni da raccogliere a fronte delle richieste e modellare le unità funzionali secondo tale criterio (vedi figura B.2.b).

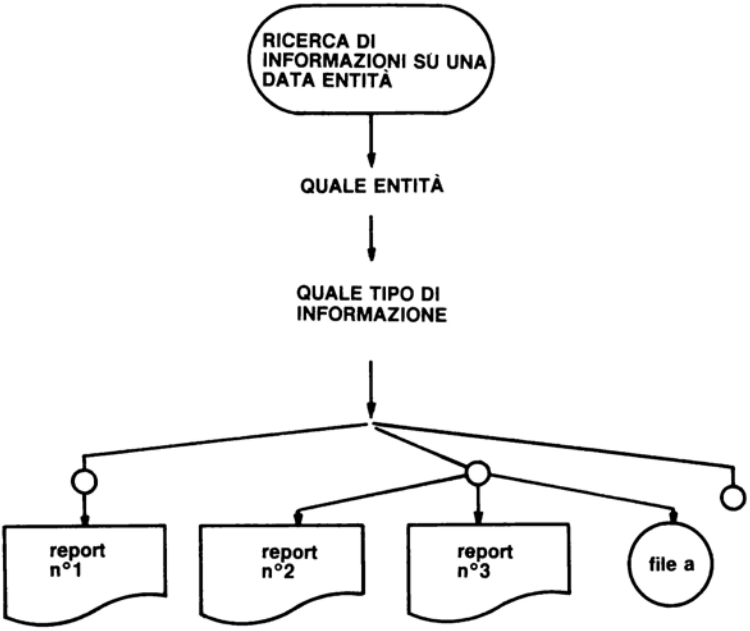


Figura B.2.b — È bene effettuare una suddivisione delle varie funzioni in attività semplici fino a ridurle in unità funzionali facilmente comprensibili ed eseguibili. I vari tipi di informazione vanno assegnati ad uno o più supporti, i più adatti all'utilizzo che successivamente ne verrà fatto. I circoletti del disegno rappresentano l'avvenuta individuazione di una unità funzionale.

Bisogna anche puntualizzare che, se si procede in questo modo (magari seguendo un processo di definizione di tipo *top-down*), si ottengono due grossi vantaggi: il primo, ovvio, consiste nel non correre il rischio di ridondanze (riguardo alle stesse attività), il secondo nel rendere assai più facile tutto quanto seguirà in questa stessa fase di pianificazione.

Tutto ciò, si era iniziato col dire, costituisce il primo compito da affrontare nel lavoro di pianificazione. Ne esiste un secondo, anch'esso assai importante, di cui vale la pena discutere: si tratta della definizione degli standards. La si è detta importante,

ma sarebbe stato bene dire necessaria ed il perché è presto visto: le entità che popoleranno il dizionario dovranno significare la stessa cosa per aree applicative diverse e le informazioni che si otterranno alla fine dovranno essere per lo più interpretabili da chiunque. Se si preferisce si può dire che è necessario che tutto ciò che entra a far parte del dizionario sia chiaro in modo inequivocabile per tutti.

Già non si è mancato di osservare che un data-dictionary è tanto più utile quanto più è usato in modo esteso e dalle aree più diverse dell'organizzazione; va aggiunto che se da un lato esso, una volta realizzato, facilita la comunicazione e rende possibile un buon grado di coordinamento, dall'altro deve essere progettato in modo tale da rispettare tutta una serie di standard. Questa è relativa ad una vasta gamma di elementi, primi tra tutti i nomi: il progettista del dizionario dovrà stabilire nomi semplici e significativi per le varie entità singole, essi dovranno essere comprensibili da parte di chiunque ed aiutare nel riconoscimento di ridondanze. Poi dovranno essere standardizzati i nomi degli attributi e le proprietà possibili per ogni entità. Infine dovranno essere standardizzate le procedure di intervento sul dizionario, in particolare quelle relative agli output ed ai tipi di informazioni desiderate.

Certamente non si è detto poco. L'obiezione: "*Come si fa? Questo è un problema di sempre!*" è inevitabile. Ci pare inutile, a questo punto della discussione, tentare delle proposte e tanto meno dare una risposta precisa; la cosa che, invece, è interessante puntualizzare è che questo tipo di problema è primario: esso assume una importanza tutta particolare nel progettare una documentazione automatizzata assai più che nel progettare un'applicazione tradizionale.

Comunque vanno sottolineati due fatti notevoli: prima di tutto la funzione di dizionario è per sua natura centralizzata e, dunque, la standardizzazione diviene più agevole, in secondo luogo il software stesso prevede normalmente una serie di standard per certi elementi per così dire forzando la normalizzazione strutturale delle informazioni.

A questo punto è opportuno interrompere i discorsi di carattere generale: per meglio comprendere il senso di quanto fino ad ora affermato e per vederne una più significativa concretizzazione, nulla può risultare più utile dell'esame di un effettivo software di data dictionary (*).

Non si tratterà, per forza di cose, di un'analisi estremamente dettagliata, bensì di un riassunto quanto più possibile preciso e completo delle sue caratteristiche e delle funzionalità principali.

Il software in esame consiste di un certo numero di moduli atti a svolgere una vasta serie di funzioni su degli elementi normalizzati: è prevista l'esistenza di una serie di tipi entità standard capace di soddisfare praticamente tutte le esigenze (vedi figura B.2.c), sono previsti numerosi tipi standard per gli attributi delle entità singole ed infine è prevista un'ampia gamma di connessioni tra queste ultime (e ciò in maniera rispondente alla logica dei problemi più diversi).

(*) Lo faremo soffermandoci quanto più possibile da vicino sul DD/DS (Data Dictionary Directory System) della Honeywell.

TIPI-ENTITÀ

SYSTEM
SUBSYSTEM
JOB TRANSACTION
PROGRAM TPR
PROCEDURE
SCHEMA
SUBSCHEMA
AREA FILE
DBREC SSREC RECORDS
SET REPORT
GROUP
ITEM

Figura B.2.c — In questo elenco sono presenti tutti i vari tipi-entità: essi sono ordinati dall'alto in basso in modo da evidenziare una gerarchia (ad un job possono appartenere più programmi, ad un file più records e così via) che avrà una certa influenza sulle connessioni che potranno essere poi stabilite (più programmi potranno essere connessi allo stesso job...).

Le funzionalità principali sono (vedi figura B.2.d): possibilità di input al dizionario *manuale* od *automatica*, facoltà di aggiornamento interattivo *on-line*, ancora facoltà di preparazione di procedure di interrogazione *ad hoc* per le richieste non previste o non prevedibili, ampia serie di output con informazioni di *cross reference* complete e generali, possibilità di generazione di sorgenti Cobol per quanto riguarda le strutture dati o di sorgenti descrittivi dei Data Base.

Accanto a queste andrebbero enumerate altre funzionalità non meno importanti, quali, ad esempio, le possibilità di protezione e le procedure di amministrazione dell'archivio-dizionario stesso; non ci soffermeremo su queste ultime, meritando già abbondante spazio una più dettagliata diesamina delle precedenti, tuttavia va ribadito che ciò che segue è la puntualizzazione di quanto, in base all'esperienza dell'autore, è più efficace ed utile.

Le varie funzioni, ed i moduli che le svolgono, fanno riferimento ad un archivio integrato. In altre parole, il dizionario vero e proprio consiste in un Data Base in cui sono immagazzinate, nella logica esposta nel paragrafo precedente, le informazioni

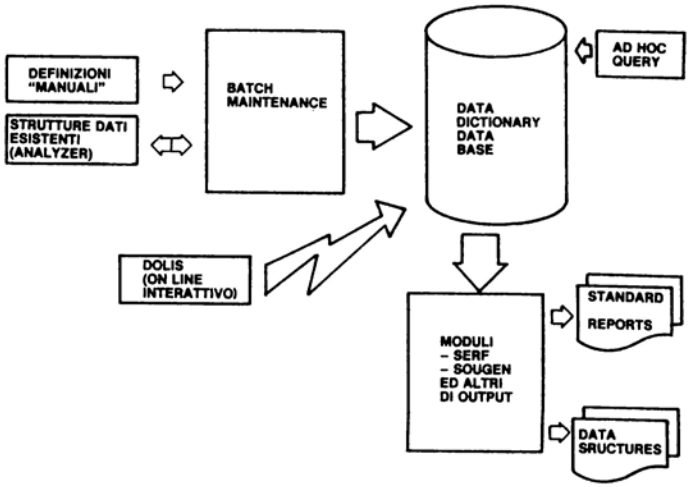


Figura B.2.d — Un primo schema di struttura del software di data - dictionary DD/DS dal quale risultano evidenti le funzionalità principali.

documentative, ossia le entità ed i loro attributi, sotto forma di tipi record diversi; le connessioni sono realizzate con opportune strutture reticolari (record connettore). Tale archivio è personalizzabile almeno per quanto riguarda le dimensioni: in base alle definizioni di quantità ottenute in fase di *planning* è possibile strutturarlo rendendolo meno (o più) voluminoso.

La fase di installazione procede attraverso alcuni passi: dimensionamento, creazione della libreria che conterrà i moduli previsti dal software e che svolgeranno le varie funzioni, creazione effettiva dell'archivio, assegnazione delle *permission* di accesso.

Ora il dizionario è pronto per essere *popolato* e si procederà alle varie fasi di input.

È a questo punto che possiamo esaminare le prime funzioni, quelle che abbiamo segnalato tra le più importanti: input *manuale* od *automatico*. Nel caso di input manuale si procede alla scrittura di un sorgente in un opportuno linguaggio, in esso verrà specificato tutto quanto vogliamo inserire e le caratteristiche (gli attributi) che

dovranno avere gli elementi descritti. Ad esempio sarà possibile specificare che va fatta la scrittura (STORE) della singola entità con il suo nome e con tutti gli attributi del caso. Poi sarà possibile connetterla ad altre entità, precedentemente inserite, attraverso l'opportuna indicazione (CONNECT) ed assegnare anche a tale connessione degli attributi. Così, per scendere maggiormente in dettaglio ed esaminare un caso più concreto, se dovessimo inserire il programma PIPPO, dovremmo scrivere:

```
STORE PROGRAM
  NAME IS PIPPO 01
  TITLE "PROG. PIPPO PROCEDURA MAGAZZ."
  STATUS IS DEVELOPMENT
  AUTHOR IS "ROSSI"
  LANGUAGE IS CBL74
  NARRATIVE "PROG. CODIFICATO IL ... PER ..."
  LINES-OF-CODE 1522
```

come si potrà notare, accanto al nome vengono comunicati vari attributi (anzi il nome stesso è inteso come tale); nel nostro caso abbiamo segnalato:

- una "*versione*" (il numero 01): si tratta di un attributo molto importante che permette di mantenere memorizzate versioni differenti della stessa entità singola: (fino a 99) ad esempio, dopo una certa serie di modifiche, si potrà fare la STORE del programma PIPPO con gli stessi attributi o con altri diversi senza per questo perdere la situazione precedente;

- il TITLE: ovvero un riassunto delle caratteristiche dell'entità, o, se si preferisce, una più diffusa spiegazione del nome assegnato all'entità;

- lo STATUS: nel nostro caso è DEVELOPMENT e ciò significa che il programma si trova in via di sviluppo, sarebbe stato possibile dichiararlo in fase di TEST, PRODUCTION, INACTIVE o PROPOSED;

- AUTHOR ci ha permesso di specificare il nome dell'autore;

- LANGUAGE ci ha permesso di definire quale linguaggio è stato utilizzato per la codifica;

- NARRATIVE ci ha permesso l'inserimento di un testo commentativo del tutto libero;

- infine LINES-OF-CODE ci ha consentito di memorizzare la lunghezza del programma in termini di linee di codifica.

Si è esaminato, è bene ribadirlo, un esempio assai particolare: gli attributi che è possibile assegnare ad una entità singola sono assai più numerosi; ad esempio, attraverso l'attributo MEMORY, avremmo potuto specificare l'occupazione di memoria del programma, con altri attributi avremmo potuto definire una categoria, una re-

sponsabilità e via dicendo. Insomma l'elenco degli attributi possibili per ogni entità singola è piuttosto lungo, soprattutto è in grado di soddisfare esigenze di tipo assai vario.

Dopo aver immesso nel dizionario le varie entità singole, si può procedere alle connessioni necessarie. Si supponga di voler connettere il file FILE-A al programma PIPPO che lo utilizza, si scriverà:

```
CONNECT XREF
 PROGRAM PIPPO 01
 TO FILE FILE-A
 USAGE-MODE IS OUTPUT
```

come si noterà, in questo caso, l'attributo di connessione utilizzato è servito a stabilire che il file viene allocato in output.

Esistono altre direttive, oltre a quelle fin'ora viste negli esempi; l'elenco completo è il seguente:

STORE
CONNECT
DISCONNECT
MODIFY
DELETE
COPY

oltre a quanto visto, cioè, si rende possibile eseguire delle sconnessioni, delle modifiche degli attributi, delle cancellazioni delle entità singole o delle duplicazioni delle stesse (cambiando il numero di versione). L'insieme di tali verbi, assieme ai formati di descrizione degli attributi, costituisce il linguaggio di *batch population* del dizionario che prende nome *Batch Syntax*.

Una volta descritto, a questo modo, tutto ciò che deve essere presente nel nostro dizionario, verrà mandato in esecuzione uno dei moduli: il MAINT (che sta per *batch MAINTenance*) in grado di eseguire la effettiva *population*, o l'input del dizionario.

È importante notare, a questo punto, che la generazione del linguaggio di controllo che permette il lancio di questo modulo (ma ciò vale anche per gli altri) è automatizzata ed assegnata ad un programma interattivo di facile utilizzo che, attraverso una successione di domande e risposte, genera il *job* necessario. Tale programma prende nome DDDJCL e permette una facile produzione di tutte le *facility* ivi comprese quelle di amministrazione e gestione.

È ovviamente assai utile disporre della possibilità di *batch population* sopra descritta, anzi essa è una necessità imprescindibile: soprattutto non deve spaventare l'idea che una scrittura della *Batch Syntax*, per così dire, manuale sia troppo lunga e

dispendiosa. L'obiezione che sorge normalmente a questo punto è che il dizionario, per essere riempito, richieda uno sforzo ed un dispendio di risorse eccessivi. In realtà questo metodo non è che uno dei possibili, anzi ne esistono, ed ora è bene subito esaminarli, alcuni diversi e completamente automatici. Il primo che ci pare valga la pena di nominare è il *Cobol Analyzer*. Si tratta di un modulo (vedi figura B.2.e) che prendendo in input i sorgenti dei programmi Cobol genera la *Batch Syntax* necessaria alle STORE ed alle CONNECT delle varie entità presenti in quel programma secondo criteri di indiscutibile logicità.

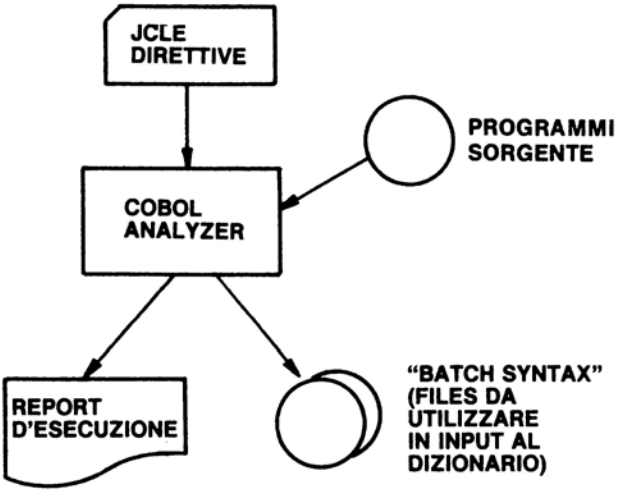


Figura B.2.e — Il modulo di analisi dei sorgenti COBOL prende in input i programmi, ne legge le strutture dati e genera la *Batch Syntax* di *population* del dizionario.

Un secondo modulo che svolge funzioni analoghe, anche se con modalità diverse, è il DIMES: anch'esso genera la *Batch Syntax* partendo dai sorgenti Cobol o dalle descrizioni di un Data Base (schema logico, schema fisico, subschema ecc.). Una volta generata in questo modo la sintassi di *population*, ancora si tratterà di lanciare il MAINT, ovvero di eseguire l'effettivo input al dizionario.

Finalmente una volta popolato il dizionario, si dovrà badare a come possano essere ottenute le varie informazioni di output. Limitando, per ora, il discorso alle funzionalità *batch*, vale la pena segnalare almeno alcuni dei moduli che le producono:

SERF (Standard Entity Reporting Feature)
SOUGEN (SOUrce GENerator)
IMPACT ANALYSIS
KEYWORD IN CONTENT.

Il primo modulo, SERF, permette di ottenere delle report circa i vari tipi di entità, per ognuno di essi è possibile generare delle informazioni di tre generi: di dettaglio, di cross-reference, riassuntive. Il modo di ottenerle è assai semplice e permette la definizione di una serie di attributi in base ai quali selezionare le entità da mettere in output. La report di dettaglio fornisce tutte le informazioni presenti nel dizionario sulle entità di un certo tipo e che abbiano gli attributi richiesti. Quella di cross-reference mette in evidenza tutte le connessioni prestabilite. Quella riassuntiva evidenzia le entità con i loro nomi ed i *title*, nonchè le date di creazione e modifica.

Se questo primo modulo permette la generazione di una serie di informazioni di tipo documentativo, il secondo, SOUGEN (vedi figura B.2.f), fornisce invece delle descrizioni attivamente poi utilizzabili. In particolare permette di ottenere, combinando tra loro le varie entità relative ai dati, le strutture degli stessi da inserire nei programmi e, al solito, le descrizioni di schema e subschema. Questa possibilità di ottenere dei sorgenti è, come si può facilmente immaginare, una delle più notevoli *facility* di standardizzazione intervenendo proprio nell'area in cui ciò risulta tradizionalmente più difficile. La maniera di ottenere tali parti di sorgente è, al solito, assai semplice ed affidata ad un'attività che permette di selezionare in dettaglio ciò che si desidera ottenere.

Il terzo modulo, IMPACT ANALYSIS, costituisce un'altra delle funzionalità più importanti (vedi figura B.2.g): esso permette di ottenere l'elenco di tutte le entità connesse ad un certo dato. Ciò significa che, preso in esame un singolo dato, ad esempio il campo di un record, è possibile sapere quali entità siano ad esso correlate (o connesse) e dunque coinvolte in una sua eventuale modifica. La report, ormai è inutile dirlo, viene ottenuta con il lancio di un'attività di tipo assai semplice in cui si specifica l'entità in questione. Si ottengono le indicazioni relative ad un *direct impact*, ovvero l'elenco delle altre entità cui quella è direttamente collegata in modo dipendente, e quelle relative ad un *indirect impact*, ovvero l'elenco delle entità che da quella dipendono.

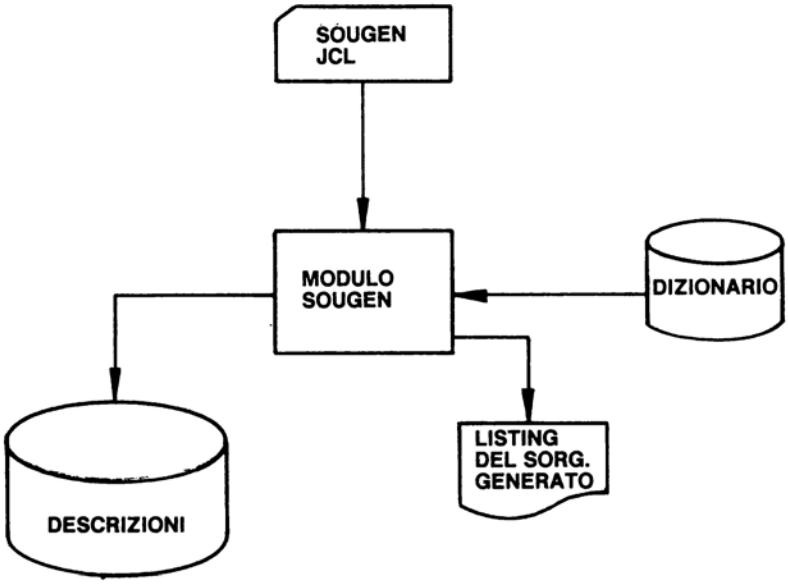


Figura B.2.f — Per mezzo di questo modulo è possibile ottenere dal dizionario la struttura dei sorgenti per quanto riguarda le descrizioni dei dati, degli schema e subschema.

L'ultima funzionalità cui si voleva accennare, KEYWORD IN CONTENT, costituisce un'altra notevole *facility*; essa permette di ottenere l'elenco di tutte le entità che, indipendentemente dal tipo, contengano nel *title* una certa parola chiave. Nell'attività di lancio si specifica tale parola: nella report di output si potranno leggere, per ogni tipo entità, i vari nomi delle entità singole nel cui titolo compare quella parola, nonchè il titolo per esteso.

Abbiamo detto che quelle succitate sono le funzionalità di reporting più importanti; certamente ne esistono altre che, se non vale la pena esaminare per esteso, con-

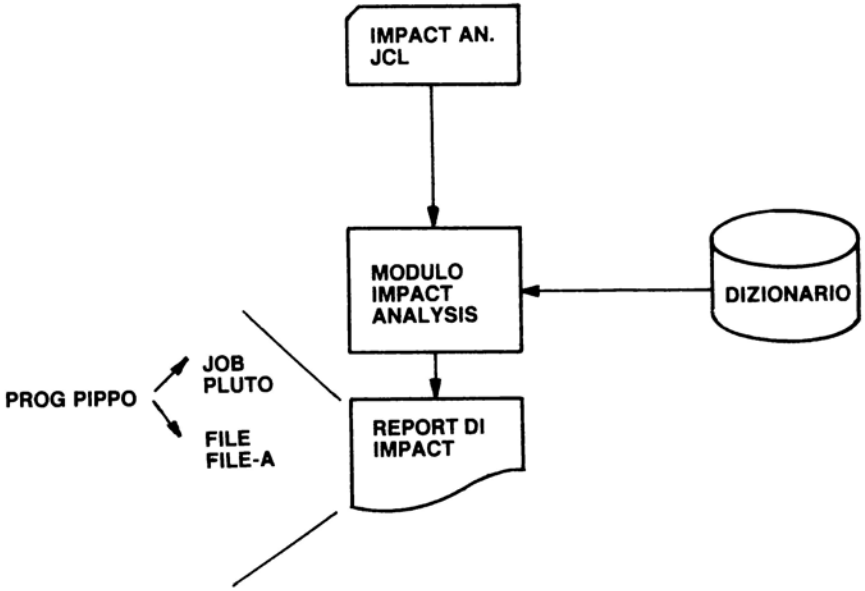


Figura B.2.g — Il modulo IMPACT ANALYSIS fornisce le utili indicazioni di "*impatto diretto e indiretto*" della modifica di una certa entità (ad esempio il programma PIPPO) su altre ad esso correlate. Ciò è ottenuto in base alle connessioni stabilite nel dizionario (come per il job PLUTO ed il file FILE-A).

viene però almeno nominare: prima di tutto esiste la possibilità di lavorare con funzionalità end-user, poi esistono le possibilità di output relative esclusivamente ad entità particolari, infine esistono tutte le funzionalità di amministrazione del dizionario visto come archivio integrato.

A questo punto, tuttavia, è bene passare ad analizzare una modalità operativa diversa: il DOLIS. Si tratta di una funzione di tempo reale che permette l'interrogazione e l'aggiornamento del dizionario in maniera immediata ed interattiva. Le possibilità offerte dal DOLIS sono ottenute in otto diverse transazioni, o funzioni.

Esse sono:

(1) — Store di un'entità

(2) — Modify di un'entità

(3) — Cross-Reference

(4) — Lista di distribuzione

(5) — Creazione di una descrizione record

(6) — Delete di un'entità

(7) — Query su un'entità.

All'atto del collegamento, sul terminale compare una maschera che permette la scelta della transazione richiesta (vedi figura B.2.h), lo svolgimento stesso di quest'ultima procede attraverso una successione di maschere facilmente interpretabili e che evitano la possibilità di errori grossolani.

```
                    DO/OS SELECTION MENU'

FUNCTION
CODE                    TRANSACTIONS

(01)......STRENT  --  STORE ENTITIES
(02)......MODENT  --  MODIFY AN EXISTING ENTITY
(03)......PRXREF  --  PROCESS CROSS REFERENCES
(04)......PRDIST  --  PROCESS DISTRIBUTION LIST
(05)......PRACCE  --  PROCESS ACCESS CONTROL LIST
(06)......CRTRCD  --  CREATE A RECORD DESCRIPTION
(07)......DLTENT  --  DELETE AN ENTITY
(08)......QRYENT  --  QUERY FOR AN ENTITY


(--)......SELECT THE FUNCTION CODE FROM ABOVE.

-------------------------------------------------------------
```

La transazione (1) permette l'inserimento della singola entità con tutti i suoi attributi; la (2) permette la modifica della singola entità: in particolare permette l'inserimento, la cancellazione, la modifica dei suoi attributi; la (3) permette di stabilire delle connessioni o di rimuoverne di precedentemente esistenti; la (4) permette di sapere a quale persona o funzione aziendale fanno capo le responsabilità delle varie entità; la (5) permette di conoscere i *permessi* di accesso; la (6), che è un analogo della *Source Generation*, produce le descrizioni dei record in formato sorgente; la (7) permette di procedere alla cancellazione di determinate entità singole ed, infine, per mezzo della (8) si possono ottenere le varie informazioni riassuntive (caratteristiche ed attributi) delle entità singole.

A conclusione di questa nostra discussione va detto ancora una volta che si è cercato di presentare una panoramica degli aspetti e delle funzionalità più importanti di uno specifico dizionario, se non del tutto completa certamente, però, abbastanza circostanziata. Ancora va sottolineato che, come ora si può facilmente comprendere, un utilizzo appropriato di questo strumento è in grado di produrre un impulso notevole verso la soluzione di quei problemi organizzativi, di standardizzazione di documentazione di cui si è dato qualche cenno.

Figura B.2.h — Al richiamo del modulo interattivo (il DOLIS) compare sul video del terminale una maschera esplicativa che ha le funzioni di "*menù*" e permette di scegliere tra otto transazioni. Queste stesse procederanno, in funzione delle diverse richieste dell'utente, con la presentazione in sequenza di una serie di altre maschere altrettanto esplicative ed in grado di guidare l'elaborazione in modo assai semplice.

INDICE ANALITICO

Administration, 127
Algoritmo di randomizzazione, 3
Analisi, Utility di, 136
Area, 117, 115, 118
Batch, 140
Capacità informativa, 11
Caricamento, 136
Catena, 48
Centralizzato, Data Base, 41-44
Ciclica, struttura, 93
CONNECT, 135
Connettore, record, 69
Data Manipulation Language (DML), 132
DBMS, 17
DDL, 113, 121
Descrizione fisica, 112
Descrizione logica, 111
DISCONNECT, 135
Disegno, 106
Distribuito, Data Base, 41-44
Dizionario dei Dati, 159 (app. B)
DMCL, 116
Documentazione, 159 (app. B)
Entry di Schema DDL, 114
Entry di Schema DMCL, 117
Entry di Subschema, 122, 124
ERASE, 135
Estensioni delle strutture, 100
FIND, 134
Gerarchica, struttura, 53
Gerarchico, modello, 21
GET, 134
Indexed Sequential, 5-10
Indipendenza, 16, 20
INSERTION, 115
Integrata, organizzazione, 45
Legami logici, 11
LOCATION, 115
Longhand, 48
Member, 48,25
Member-member, struttura, 87
Modalità operative, 140-145
Modelli di Data Base, 21-40
MODIFY, 135
Organizzazione integrata, 45
Organizzazione logica e fisica, 18
Organizzazioni, 3, 5
Owner, 48, 25
PAGE-INTERVAL, 115
PAGE-SIZE, 115
Pointer, 48, 25
Privacy, 115
PRIVACY, 130, 115
Query, 141
Raggruppamenti, 64, 106-110
Randomizzazione, 3
READY, 134
Record, 48
Relazionale, modello, 28
Relazioni logiche, 11, 21
Rete di Data Base, 41-44
Reticolare doppia, 78-96
Reticolare, modello, 25
Reticolare, struttura, 69, 79, 87
Scaricamento, 136
Schema, 111
Schema DDL, 113
Schema DMCL, 116
Schema fisico, 116
Schema logico, 113
Set, 48
Set opzionale, 93
Shorthand, 48
Software di gestione (DBMS), 17
STORE, 135
Struttura ad albero, 62
Struttura ciclica, 93
Struttura gerarchica, 53
Struttura member-member, 87
Struttura reticolare, 69, 79, 87
Subschema COBOL, 122
Subschema FORTRAN, 124
Subschema, scelta dei, 147 (app. A)
Subschema, 119
Tempo reale, 140
Utilities, 135
Validazione, 119-126

Il libro, destinato a utenti e responsabili di archivi tradizionali, insegna che cos'è un Data Base, che cosa bisogna sapere per utilizzarlo e come lo si usa.
Dopo una rassegna dei diversi modelli di Data Base passa a descrivere il Data Base integrato. In un secondo momento affronta le problematiche relative al progetto e alla realizzazione di un Data Base e infine presenta i linguaggi di descrizione.
L'autore, Roberto Doretti, si è laureato in Fisica all'Università degli Studi di Milano. Ha iniziato la sua attività direttamente al centro Formazione della Honeywell I.S.I. ed ivi opera attualmente occupandosi dei corsi sui grandi sistemi.

L. 22.500 Cod. 526P ISBN 88-7056-174-7

176

DATA BASE

Roberto Doretti

GRUPPO EDITORIALE JACKSON